A Tile Assembly Model With Hexagon Shaped Tiles

by

Andrew G. Sinclair

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
January 2015

Thesis advisor
Michael Domaratzki

Author
Andrew G. Sinclair

A Tile Assembly Model With Hexagon Shaped Tiles

Abstract

The field of nanotechnology has enabled scientists to perform fascinating engineering manipulations of biological substrates. Systems of DNA are now able to perform algorithmic computations by way of constructing biological modules composed of DNA macromolecules and using laboratory techniques available to biological sciences. The tile assembly model is an established model of biomolecular computing: using properties of DNA macromolecules to define constructions of self-assembling biological systems. The existing tile assembly model uses the concept of DNA tiles conceptually shaped as squares and exposes the tiles to carefully controlled biological conditions. The result is that under this process we can design and create these systems to compute solutions to algorithmic problems. Hexagons are the only two-dimensional regular polygon other than squares that can tile a plane infinitely leaving no space uncovered, where only translations of the initial polygon is allowed. Therefore hexagon-shaped DNA tiles can be defined to cover a planar surface, with the notable difference of six adjacent tiles per position versus the four adjacent neighbours in traditional four sided tiles.

In this thesis, we will define a generalization of the tile assembly model that supports six-sided DNA tiles, in addition to the traditional four sides. We will introduce

a problem known as the 0-1 Knapsack problem that is currently unsolved with square tiles. Moreover, a solution to the problem was attempted by tile assembly model researchers, however we show there is an error in their solution. After we analyze their solution and discover the shortcomings of square tiles under those constraints, we then show this fault is not applicable to hexagon tiles. Therefore, we show that the 0-1 Knapsack problem is solvable using hexagon shaped tiles.

Contents

List of Figures

Acknowledgments

There are numerous people to whom I must show my sincerest gratitude. Without the help, support, and encouragement of these individuals, I would not be able to present to you this document. More importantly, I would not be able to show for my efforts the pride and accomplishment that comes with attaining a goal as challenging as this one. To my parents, Evelyn and Michael, thank you for the motivation and support you have provided since the beginning. No one showed more anticipation from the moment I first told you my intention to complete my Masters degree. To my wife, Stephanie, you continued to show me unfathomably deep understanding, love, and support no matter how distant my goals would appear. And finally I wish to show my appreciation to no end to my supervisor, Dr. Michael Domaratzki. The countless hours you have poured selflessly into my education and success towards this endeavour have far outweighed my most far-fetched expectations of what assistance I would be afforded upon beginning this journey. Countless forms, referrals, edits, courses, lessons, advice... and the list goes on. Not only that, but the amount of trust and respect you have shown myself in my situation, allowing me the chance to resume my studies and depend on your unerringly high degree of assistance after the length of silence in our correspondence, has effected from me my respect, in the sincerely, most heartfelt meaning of the word.

To all, not only could I not have succeeded without you, I would not have known where to begin! Thank you.

This thesis is dedicated to all the loved ones who were with me on this journey. Poe, Oma, Aunt Linda, Onkle Tony, Uncle Eddy, Tante Moni, Uncle Kelber, Auntie Yvonne, Peter, Greg, and Mei.

Chapter 1

Introduction

1.1 DNA Tile Assembly Model

The tile assembly model is an established model of Biomolecular Computing: using properties of DNA macromolecules to define self-assembling biological systems. We can create these systems to compute solutions to algorithmic problems. The tile assembly model has a geometric analogue as Wang tiles: two-dimensional square tiles with colour coding on their edges. Wang tiles are a theoretical geometric construction designed to tile adjacent to one another under the condition that abutting edges must match in colour. Barring rotation of tiles, hexagons are the only other regular polygon that can tile a plane indefinitely leaving no space uncovered. Therefore hexagon tiles can be defined to cover a plane of infinite size, in the same way as square Wang tiles, with the notable difference of six adjacent tiles per position versus the four adjacent neighbours in traditional four sided Wang tiles.

1.2 Motivation

The reason for the six-sided tile assembly model is to investigate how the added connectivity can simplify certain flows and thereby simplify the solutions to problems. Chapter 6 introduces several hexagon solutions for common arithmetic operations that have well known square-tile solutions in current literature. In the examples provided, the hexagon tiles are more legible and less complex in the tile defintions.

While we do not define an exact criterion to determine whether hexagons will improve any given square solution, we will demonstrate a situation where square tiles fail to provide a completely satisfying solution to the 0-1 Knapsack Problem, a well-known NP-Complete problem. In response, Chapter 7 will show a hexagon solution to the same problem that:

a. satisfies it,

b. improves legibility, and

c. uses less complexity in the definitions.

We will demonstrate that hexagon solutions can have an advantage over the established square tile solutions for many common problems already attempted in the literature. As research continues in using the tile assembly model to solve algorithms, it's reasonable to consider that the hexagon tiles will have an advantage of being the preferred representation. In short, this thesis will show that the hexagon tiles are more likely to succeed and do so with less resources.

1.3 Roadmap

Chapter 2 will start the discussion by introducing the definitions and core concepts required to reason about tile assembly model and its application. The hexagon definitions will be developed as an extension and generalization of the existing literature's square tile assembly model. Additional concepts for describing the tile assembly model systems and their applications are also defined in this chapter.

Chapter 3 introduces the reader to background material for the tile assembly model research and its precursory studies so as to support the concepts used in this thesis. Related materials are also discussed in this chapter to provide clear boundaries for what facts are related to the tile assembly model.

Chapter 4 defines 0-1 Knapsack, an NP-Complete problem. We present the current solution using the existing four-sided tile assembly model techniques. We then show that this solution has a major shortcoming which hinders the square model's usefulness in solving the 0-1 Knapsack problem.

Chapter 5 defines a scenario that occurs with square tile assembly model techniques in the current literature. We provide a proof of a particular unsatisfactory detail of the best known four-sided solution which, despite best efforts, cannot solve 0-1 Knapsack. The proof shows why the current square tile techniques are unable to solve the 0-1 Knapsack problem.

The results from Chapter 5 motivate the need to explore additional techniques, namely the six-sided tile assembly model. In Chapter 6, we will define various six-sided tile assembly model solutions that demonstrate the fundamental techniques shared by both square and hexagon models. We give example usages of six-sided tile

assembly model systems to solve simpler problems that are well known, and have established solutions, in the literature for square tiles. The goal in this section is three-fold:

a. to establish a working six-sided tile assembly model theory for fundamental problems in the field,

b. to highlight intermediary results that show advantages to using hexagons, and

c. to familiarize the reader with common themes and concrete tile definitions that will be presented as essential building blocks of the tile assembly model — and in particular six-sided tile assembly model — solution defined in the next chapter.

In the first half of Chapter 7, we define the hexagon tiles we have used to solve the 0-1 Knapsack problem with the six-sided tile assembly model. We provide figures and examples for the tiles in order to explain their use and relation to one another in the solution for 0-1 Knapsack. In the second half, we define the process for computing the solution to 0-1 Knapsack from Chapter 7 with those six-sided tiles and how to interpret the results. By nature of the tile assembly model, many outcomes are possible due to non-deterministic choices in the algorithm made at the biological level. Massively parallel executions of the computational DNA self-assembly process ensures an aggregate result consistent with the expected solution of the 0-1 Knapsack problem. Therefore we enumerate the outcomes of such an execution and define how to ascertain the final result.

Finally, in Chapter 8 we summarize the results achieved in this thesis. We show the advantage to using the six-sided tile assembly model over the four-sided tile assembly

model in the general scenario.

Chapter 2

Definitions

2.1 Tile Assembly Model for degrees $n = 4, 6$

The Tile Assembly Model (TAM) is a model of computation that combines properties of DNA self-assembly and geometry to "build" computations out of regular polygons. The tile assembly model is well defined in the work by Wang [15] and others, but those definitions are for four-sided (i.e., square tile) systems. For the work in this thesis, we will be using more generalized notations. In particular we wish to define the novel idea of hexagon tile systems, therefore six-sided tiles must be considered. What follows are the definitions for the n-tile assembly model, where $n \in \mathbb{N}$, with particular emphasis on six-sided tiles and how they differ from the four-sided tiles.

An n-tile $\boldsymbol{t}$ is an ordered tuple of length $n \in \mathbb{N}$. The set of all n-tiles is denoted T_n. The set Σ is known as the set of binding domains. Each element of a tile $\boldsymbol{t}$ is a member of the set of labels Σ which includes the null string denoted NULL and

the empty string denoted BLANK. Throughout this thesis, we will illustrate tiles with NULL as a thick, black line on a tile and BLANK as a thin, black line on a tile. BLANK will never have a symbol in illustrations, however it is sometimes for convenience that we illustrate the NULL edge with the symbol $\varnothing$.

As a naming convention, the cardinal directions and their subdivisions are used to map elements from $\boldsymbol{t}$ to unique identifiers. These identifiers are known as the directions and are denoted by an n-tuple $\boldsymbol{D}_n$. The exact mapping used for a given n-tile is an arbitrary choice and therefore without loss of generality this thesis will use the following conventions for $n = 4$ and $n = 6$ tiles: $\boldsymbol{D}_4 = \langle N, E, S, W \rangle$ and $\boldsymbol{D}_6 = \langle W, NW, NE, E, SE, SW \rangle$. A function is defined to return the binding domain of a tile given the tile and direction. It is denoted by $bd_d : T_n \mapsto \Sigma, \forall d \in \boldsymbol{D}_n$. The empty tile is an ordered tuple of length n given as EMPTY $= \langle \text{NULL}, \text{NULL}, \ldots, \text{NULL} \rangle \in T_n$.

Occasionally, it is convenient to denote a tile and its mirror image. We define the mirror image of a 6-sided tile $\boldsymbol{t} = \langle u, v, w, x, y, z \rangle \in T_6$ as $\bar{\boldsymbol{t}} = \langle x, w, v, u, z, y \rangle$, where $u, v, w, x, y, z \in \Sigma$.

A position is an element of $\mathbb{Z}^2$. The neighbour function $\text{Adj}_d : \mathbb{Z}^2 \mapsto \mathbb{Z}^2$ gives the adjacent position, in the specified direction, d. Two positions (x, y) and (x', y') are said to be neighbours if and only if there exists a $d \in \boldsymbol{D}_n$ such that $\text{Adj}_d(x, y) = (x', y')$.

Adj_d is defined explicitly for each $d \in \boldsymbol{D}_n$ depending on the system. The current research on systems with $n = 4$, where $\boldsymbol{D}_4 = \langle N, E, S, W \rangle$, define $\text{Adj}_N(x, y) = (x, y+1)$, $\text{Adj}_E(x, y) = (x+1, y)$, $\text{Adj}_S(x, y) = (x, y-1)$, and $\text{Adj}_W(x, y) = (x-1, y)$.

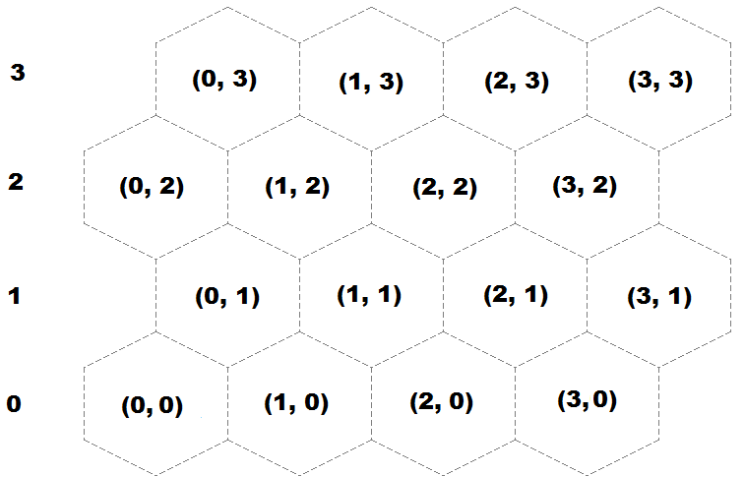

Figure 2.1: Hexagonal Grid Indexing.

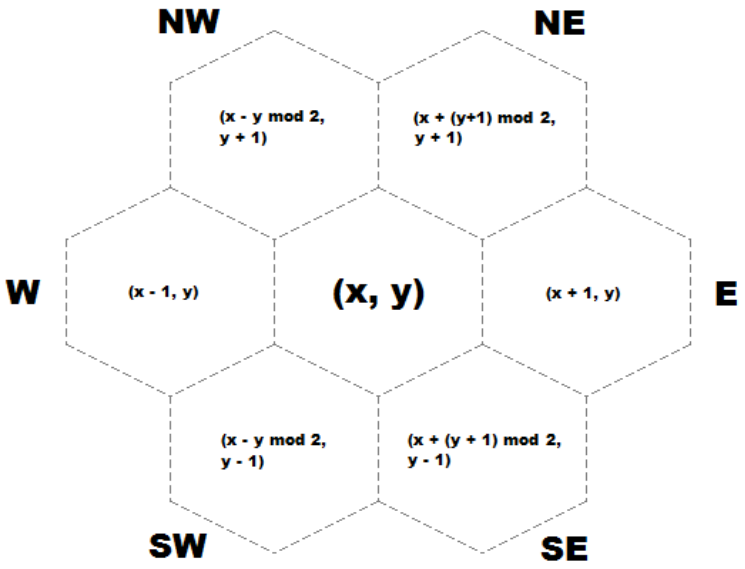

Figure 2.2: Hexagonal Neighbour Function.

Such a mapping corresponds to adjacent squares on a planar cartesian grid. Therefore using similar motivation, we define for $n = 6$, where $\boldsymbol{D}_6 = \langle W, NW, NE, E, SE, SW \rangle$, a mapping to represent adjacent hexagons on a planar hexagonal grid. Numerous methods for indexing cells in a hexagonal grid exist, therefore the following is only one such method: $\text{Adj}_W(x, y) = (x - 1, y)$, $\text{Adj}_{NW}(x, y) = (x - y \bmod 2, y + 1)$, $\text{Adj}_{NE}(x, y) = (x + (y + 1) \bmod 2, y + 1)$, $\text{Adj}_E(x, y) = (x + 1, y)$, $\text{Adj}_{SE}(x, y) = (x + (y + 1) \bmod 2, y - 1)$, and $\text{Adj}_{SW}(x, y) = (x - y \bmod 2, y - 1)$. Figure 2.1 illustrates the relation between these functions. Figure 2.2 shows the indexing on a larger subsection of grid. Observe the rows of contiguous cells adjacent in the West-to-East direction. Likewise, the y value in the (x, y) tuple remains constant along contiguous cells in each such row.

For all $d \in \boldsymbol{D}_n$, Adj_d has an inverse. The inverse is denoted Adj_d^{-1} and two neighbour functions Adj_{d_1} and Adj_{d_2} are inverses if $\text{Adj}_{d_1}(x, y) = (x', y') \Leftrightarrow \text{Adj}_{d_2}(x', y') = (x, y)$, for all (x', y'), $(x, y) \in \mathbb{Z}^2$. As a notational convention, we denote $d^{-1} = d' \in \boldsymbol{D}_n \Leftrightarrow \text{Adj}_d^{-1} \equiv \text{Adj}_{d'}$. Therefore, we can refer to the inverse direction, $d \in \boldsymbol{D}_n$ in the same way we consider inverse positions along the same direction d. Explicit to the hexagon tiles, the inverse pairs $(d, d^{-1}) \in \boldsymbol{D}_n \times \boldsymbol{D}_n$ are:

- (N, S),
- (NE, SW), and
- (NW, SE).

A configuration is the unique mapping of positions to tiles $A : \mathbb{Z}^2 \mapsto T_n$. As a convention, we write $(x, y) \in A \Leftrightarrow A(x, y) \neq \text{EMPTY}$. Intuitively, A may be empty

at first and we continuously update it to represent the addition of new tiles to the system. As a metaphor, consider a jigsaw puzzle with pieces arrayed in a grid pattern. As the solution to the puzzle progresses, new pieces are added to the growing solution set.

g is the strength function $g : \Sigma^2 \mapsto \mathbb{N}$ which, intuitively, is used to denote the bonding strength between two adjacent sides. Specifically, we define $g(\sigma, \text{NULL}) = 0, \forall \sigma \in \Sigma$ and we make the assumption that $g(\sigma_1, \sigma_2) = 0$ if and only if $\sigma_1 \neq \sigma_2$. Brun [4] assumes that $\forall \sigma \neq \text{NULL}$, $g(\sigma, \sigma) = 1$ and $\forall \sigma' \neq \sigma$, $g(\sigma, \sigma') = 0$ and denotes $g = 1$ as a shorthand notation for this assumption. In the work of others [3, 4, 20, 5, 16] and this thesis included, we allow the general case where $g(\sigma, \sigma) = m$, for some $m \in \mathbb{N}$.

A tile system is defined as a 5-tuple $\mathbb{S} = \langle T_n, g, \tau, n, \boldsymbol{D}_n \rangle$, where T_n is the set of tiles in the system, g is the strength function, $\tau \in \mathbb{N}$ is the temperature which is used in defining the assembly contstraints below, n is the number of sides per tile, and $\boldsymbol{D}_n$ gives the directions.

Rothemund and Winfree [13] define a tile assembly model as having a constraint known as the temperature function. They define a parameter, $\tau \in \mathbb{N}$, and enforce the following additional rule:

> A tile $\boldsymbol{t}$ may attach to a configuration A at location (x, y) only if the number of tiles in A orthogonally adjacent to (x, y) is greater than or equal to τ.

The temperature function is so called because the amount of heat is biologically significant in the tile assembly model's biological implementation. The value of τ in the temperature function is correlated to the heat of the physical system. Essentially, the physical strands of DNA are able to attach when the temperature of the system al-

lows. In particular, double strands are attached (annealed) and detached (denatured) based on the temperature of the system [17]. The actual temperature depends on the length of strands however longer strands will always require more temperature. The process involves heating the solution until the DNA strands denature. Conversely, when the system is allowed to cool, complementary double strands will anneal.

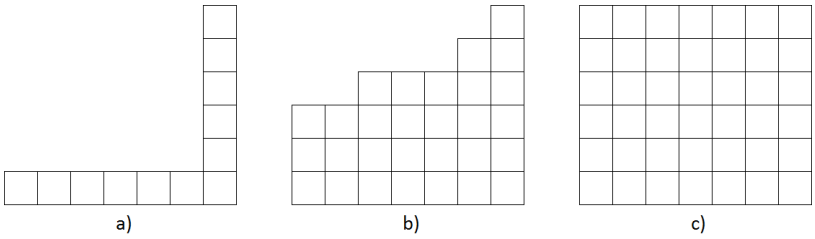

Figure 2.3: A 6×5 L-shaped starting configuration, illustrated at three steps during assembly

2.2 Assembly

In conjunction with the elements of the system $\mathbb{S}$, we now define the operation of attaching a tile to a configuration thus producing a new configuration. We define the production operation by imposing constraints on which tile $\boldsymbol{t} \in T_n$ may attach to some configuration A at a position $(x, y) \in \mathbb{Z}^2$ as follows:

$\boldsymbol{t}$ may attach to A at (x, y) producing A' if and only if:

$$(x, y) \notin A, \tag{2.1}$$

$$\sum_{d \in \boldsymbol{D}_n} g(bd_d(\boldsymbol{t}), bd_{d^{-1}}(A(\mathrm{Adj}_d(x, y)))) \geq \tau, \tag{2.2}$$

$$\forall (u, v) \in \mathbb{Z}^2, (u, v) \neq (x, y) \Rightarrow A'(u, v) = A(u, v), \tag{2.3}$$

$$A'(x, y) = \boldsymbol{t}, \tag{2.4}$$

$$\begin{gathered}\text{for all } d \in \boldsymbol{D}_n, \text{ such that } A(\mathrm{Adj}_d(x, y)) \neq \text{EMPTY}, \\ bd_d(\boldsymbol{t}) = bd_{d^{-1}}(A(\mathrm{Adj}_d(x, y))).\end{gathered} \tag{2.5}$$

These five constraints provide a formal basis for the restrictions and process of the tile assembly model. The constraints can be explained in simple terminology as follows. Equation 2.1 specifies that any new tiles can only assemble at a position (x, y) only if that position is currently empty in the configuration A. The notation of inequality 2.2 belies its simplicity. This inequality ensures that a tile can only assemble at a position if the number of all neighbouring tiles is equal to or exceeds the value τ. Next, constraints 2.3 and 2.4 define precisely the effective change from A to A'. Namely, that when a new tile is added to a position (x, y), then the new configuration A' will contain only the new tile, and the value at all other positions $(u, v) \neq (x, y)$ are left unchanged. Finally, formula 2.5 declares that for non-empty neighbours, all abutting edges must match every time a new tile is assembled to A.

As an example of these constraints, Figure 2.3 illustrates the concept of using an L-shaped starting configuration when $\tau = 2$. At step (a), only tiles from the L-shaped starting configuration are assembled. In step (b) all subsequently placed tiles will be assembled within the boundaries of the 6×5 area because of the $\tau = 2$ constraint. No gaps will be present at any time in the configuration process. Finally, step (c) illustrates the final configuration: a 6×5 tile region, bounded by the original starting tiles. Since $\tau = 2$, no more tiles may be added to anywhere at this point in the assembly, therefore this configuration is final.

Inequality 2.2 is very important and referenced frequently throughout the remain-

der of this thesis. Recall that τ is a constant value defined by the tuple $\mathbb{S}$. Simply, the inequality 2.2 states that the sum of the binding strength of all edges of a given tile t placed at position (x, y), as compared to neighbouring tiles of (x, y) currently assembled in A, must equal or exceed the temperature τ. In the frequent case when $g = 1$, the inequality simplifies to inequality 2.6.

$$|P| \geq \tau, \text{where } P = \{d \mid \forall d \in \boldsymbol{D}_n\} \tag{2.6}$$

For a complete formalization, the production of a configuration A into A' is analogous to an operation which adds tile $\boldsymbol{t}$ to A and produces A'. We denote this operation as $A \xrightarrow{\boldsymbol{t}} A'$. If conditions 2.1 and 2.2 are true in A, then A' is defined such that the third and fourth conditions are true. If the first two conditions are not met, then we let $A' = A$. If $\forall \boldsymbol{t} \in T_n$, $A \xrightarrow{\boldsymbol{t}} A'$ gives $A = A'$, then A is a final configuration. Intuitively, A is a final configuration if there exists no tile in T_n such that it fits into the configuration of A. Let $\xrightarrow{T_n^*}$ be the operation that yields the set of all configurations that are sequentially produced from A by any sequence of any length of tiles $\boldsymbol{t} \in T_n$, i.e., the transitive closure of $\rightarrow$. The transitive closure of $\rightarrow$ is defined recursively as follows.

Define variables:

- $\mathcal{A}_{A_0} = \{A_0\} = \{A\}$
- $\mathcal{A}_{A_{i+1}} = \{A' \mid A_i \xrightarrow{\boldsymbol{t}} A', \boldsymbol{t} \in T_n \text{ and } A_i \in \mathcal{A}_{A_i}\}$, for each $i > 0$.
- Then $A \xrightarrow{T_n^*} \mathcal{B}_A \Rightarrow \mathcal{B}_A = \bigcup_{i \in \mathbb{N}} \mathcal{A}_{A_i}$

The resulting set may be arbitrarily large and contain many configurations that are not final configurations. However, for some initial configuration A let $\mathcal{A}_A$ be the set of configurations produced by the transitive closure of A by elements in T_n, such that $A \xrightarrow{T_n^*} \mathcal{B}_A$. If the subset of final configurations in $\mathcal{B}_A$ has only one element, then we call that final configuration a unique final configuration. In notation, we let $\mathcal{F}_A = \{B \in \mathcal{B}_A \mid B \xrightarrow{\boldsymbol{t}} B' \Rightarrow B = B', \forall \boldsymbol{t} \in T_n\}$. $\mathcal{F}_A$ is the set of all final configurations and if $|\mathcal{F}_A| = 1$, then A is said to have a unique final configuration.

For ease of discussion, we label each tile with a symbol or value. The label will not hold mathematical meaning, however it will be used to identify a tile's representation in the configuration. For example, a contiguous row of tiles labelled with the bits 0 and 1 typically represent a binary number composed of one bit per tile. When we say that the label holds no mathematical meaning, we are saying that neither the existence nor the value or symbol of the label are necessary for any of the results in this thesis. We make use of the label to categorize tiles within the context of a proof, however those tiles remain uniquely defined were there no label to begin with. If we make reference to a tile's label, or more frequently, reference a tile by its label, it's only to identify a subset of one or more tiles defined in the theorem. For example, stating that a given position in a given configuration is a "one" is stating that the tile in such a position is an element in the set of all defined tiles with a label of "one". The application of referencing the tiles' labels in this way will facilitate intuitive discussions of the meaning of tile subsets. Results will more often make use of the formal tile definitions which do not designate any labels.

The definition for labels will be $v : T_n \mapsto \Sigma$, where T_n is the set of all tiles and Σ is

the set of binding domains. As a convention for this thesis, we state the assumption that $v(\boldsymbol{t}) \in \{\text{Adj}_d(\boldsymbol{t}) \mid d \in \boldsymbol{D}_n\}$. Namely, the label $v(\boldsymbol{t})$ for some tile $\boldsymbol{t} \in_n$ is an edge of $\boldsymbol{t}$. Therefore we can define the range and domain of the label function as $v : T_n \mapsto \{\text{Adj}_d(\boldsymbol{t}) \mid d \in \boldsymbol{D}_n,\ \boldsymbol{t} \in T_n\}$. For example, suppose the we have defined a tile as $\boldsymbol{t} = \langle 0, 1, A, B, C, D\rangle$. Any of the symbols in $\{0, 1, A, B, C, D\}$ are valid labels, therefore one example may be $v(\boldsymbol{t}) = A$.

2.3 Starting Tiles

Every instance of the Tile Assembly Model will begin with a starting configuration as in Figure 2.3. The starting configuration is defined as the initial state of A and contains tiles that are separate from T, which we denote with Γ. The starting tiles are not a part of the assembly at later stages. This fact is defined by formula 2.3, which state that all tiles in the configuration A are not modified by the assembly process. In fact, the configuration of the starting tiles are essentially the input to the algorithm, since they encode the initial state of the system in advance of the assembly process.

In a DNA implementation, there is an additional biomolecular process required by the experimenter to initialize a starting configuration with the required data. Rothemund et al. [14] created the first ever starting configuration using scaffolded strands of double crossover DNA. A single thread of DNA passes through the entire starting configuration, tying the tiles together and providing structural stability. Additional strands are introduced to the system at the same with the properties that they are attached to the structure strand and expose an additional unbounded DNA strand

known as a sticky end. These sticky ends are what constitute the valued edges exposed by the starting configuration when viewed as the tile assembly model. When the assembly of tiles from T occurs, they will initially attach to the starting configuration's sticky ends, and subsequently each other after that.

Chapter 3

Related Work

3.1 Early Models for Biomolecular Computing

Adleman [1] was the first researcher to create a biological system made of DNA for computing. Adleman developed DNA computing techniques to compute a solution to an instance of Hamiltonian Path, a well known NP-Complete problem.

The Hamiltonian Path problem consists of finding a path through a graph that visits each vertex exactly once. Adleman used the convention that a start and end vertex is given as an additional constraint to the problem. The instance of Hamiltonian Path that Adleman solved was with a 7-vertex, connected graph but other graphs would be theoretically possible. There is no specific limitation or reason for using seven vertices in his initial experiment. Using DNA, he took advantage of the Watson-Crick complementary feature of how DNA will adhere deterministically, joining two complementary single stranded DNA into one double stranded DNA helix. He begins by defining the set of DNA strands he will use to solve the problem. In-

tuitively, he creates a strand for each edge and for each vertex. Each edge strand contains half of a complement of one vertex, and half of a complement of another vertex. The two halves represent vertices in a edge denoted by symbols uv. That way, when the strands are allowed to join (through a biomolecular process known as ligation), they create a double stranded piece of DNA where one strand encodes the vertex list of the visited nodes in a candidate Hamiltonian path. The second strand reads as the complement of the list of edges making up the Hamiltonian path. Edges are defined so that the assembled double strand has no final sticky ends on either side. Therefore, combining all these partial strands in a solution generates a random candidate path, many of which are invalid under the constraints of the Hamiltonian Path solution definition. Therefore, after strands have assembled, Adleman filtered out incorrect paths. First, he extracted strands that do not contain the start and end vertices. Next, he extracted strands that do not have the correct path length. Recall that a correct Hamiltonian path for a 7-vertex graph will always visit 7 vertices therefore the correct double strand of DNA must be a specific size. Finally, Adleman identified and removed all strands that do not pass through vertex v_i for all vertices i. After the generation of single strands, the ligation into double strands, and the extraction of strands not representing a successful Hamiltonian path, any remaining strands represent the solution to the 7-vertex Hamiltonian Path problem. If there are not any strands remaining, then this represents that the initial graph did not contain a Hamiltonian Path.

Adleman's work successfully demonstrated the feasibility of DNA as a computing substrate. Solving an NP-Complete problem is a good first step toward legitimiz-

ing an algorithmic technique, however it remained to be seen what can and cannot theoretically be accomplished with DNA. Furthermore, the applications for DNA computing were brought into question after Adleman's success. Some of the positive contributions shown by Adleman's technique include the massively parallel nature of his algorithm. When single strands are generated, a biochemical technique called polymerase chain reaction (PCR) is used to replicate up to 10^{14} copies of a strand. The subsequent steps of combining double strands and filtering inappropriate solutions also occur on the order of 10^{14} operations per step. The drawbacks with Adleman's methods affect both time and space constraints. Performing the algorithm steps in a DNA substrate required considerable laboratory involvement on the researcher's behalf. The amount of time to complete the entire experiment was seven days of laboratory testing. Also, due to the randomization that occurs in the natural DNA ligation process and other laboratory operations performed by Adleman, many steps required multiple trials to reduce the chance of errors or mis-readings. The entire process was more arduous and time consuming than what we are used to in modern computing methods. Finally, the space requirement, or more practically the amount of materials required, was brought into question. Linial and Linial [11] approximate that using methods similar to Adleman's for solving Hamiltonian Path, a 70-vertex graph would require 10^{25} kilograms of DNA. In reference, this weight is roughly equivalent to that of our own planet Earth.

Adleman's work was pioneering in the practical application side of DNA Computing, however there was no prior information on the theoretical algorithmic limitations of a generalized DNA computing model. Therefore, in response to Adleman's work,

computing with DNA was shown to be computationally complete when Winfree [19] proved that DNA computations are Church-Turing universal. DNA computing models were therefore linked to the more familiar computing model of Turing Machines, cementing Adleman's ideas as feasible for algorithm design.

3.2 First Tile Assembly Model with DNA

The earliest work that showed DNA self-assembly as a feasible domain for implementing the Tile Assembly Model was in creating Sierpinski triangles. A Sierpinksi triangle is the fractal pattern that can be defined recursively with logical bits "0" and "1" over a grid row-by-row as an XOR of the two adjacent values of the previous row, i.e., $L(x, y) = \text{XOR}(L(x-1, y-1), L(x, y-1))$. Rothemund et al. [14] created Sierpinski triangles using DNA tiles and their work was essential in demonstrating that DNA self-assembly is physical domain that can be used for the computation of algorithms. They performed the following biological steps in a laboratory setting to create the Sierpinski triangles.

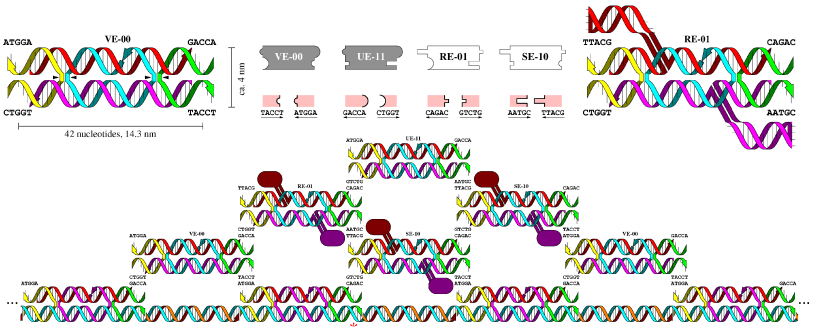

Figure 3.1: Rothemund et al.[14] illustrate the 0 and 1 tiles defined for creating the tile assembly model that generates Sierpinski triangles.

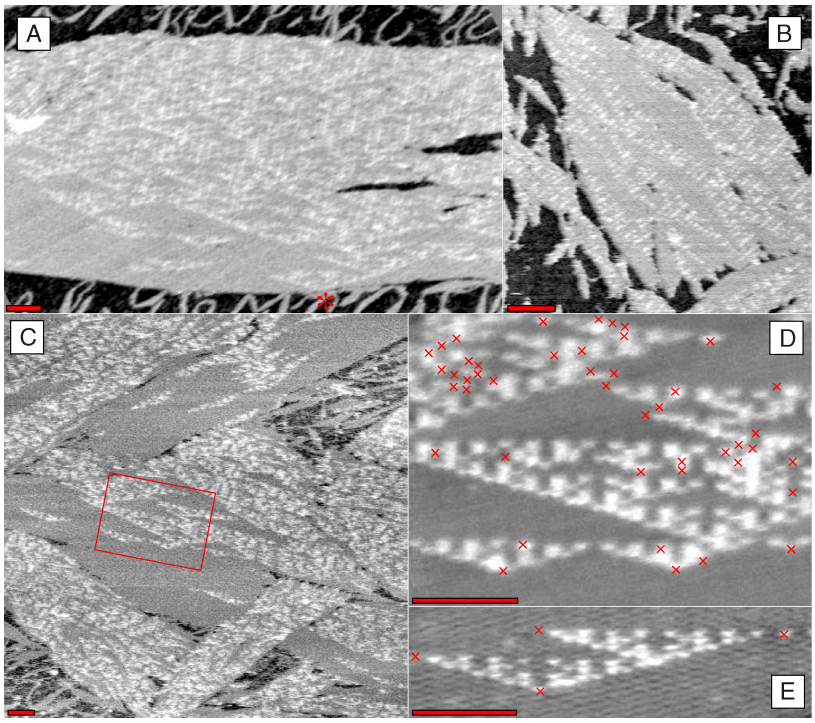

Figure 3.2: Rothemund et al.[14] use a technique known as atomic force microscopy to visualize the DNA tiles formed by the Sierpinski triangle tiles in a biological system. Section C highlights a region that assembled correctly.

The tiles were created using single strands of DNA to assemble double helix structures joined in the middle and having four sticky ends, known as double crossover strands (see Figure 3.1). The sticky ends were designated top-left, top-right, bottom-left, and bottom-right. The nucleotides in those sticky ends were defined such that top-left would combine with bottom-right and top-right with bottom left. In addition, tiles were defined as representing either 0 or 1, analagous with the interpretation of Sierpinski triangles being formed by subsequent XOR operations between neighbouring tiles defining the output of the tile in the next row and adjacent to the parent tiles. Therefore, there will be two "0" DNA tiles defined and two "1" tiles. Specifically, a "1" tile would have two parent tiles that are a "0" and "1". Conversely, one "0" tile

would have two parent tiles that would be either both 1's or both 0's.

Rothemund et al. [14] define the tiles in Figure 3.1. The tiles are labelled VE-00, UE-11, RE-01 and SE-10. Tiles VE-00 and UE-11 are "0" tiles and RE-01 and SE-10 are "1" tiles. Observe in the top right of Figure 3.1 that the "1" tiles have a pair of additional DNA "loops" that are attached. These loops are called hairpins and will be used as the distinguishing charactersitic with respects to "0" tiles, when viewed under a powerful microscope.

Given a starting strand created the same way and that seeds an initial "1" tile, all the tiles will be combined in a liquid solution along with other biological agents that enables the process. The temperature of the system is an important controlling factor that enables the assembly process. After the DNA tiles have been added to the solution, the entire system is heated above a pre-defind threshold and allowed to cool. These changes in temperature cause the DNA tiles to anneal. Recall from Chapter 2 the temperature required for annealing is correlated to the length of the DNA strands, and can vary by system. In order to verify the correctness of the assembly, the solution is deposited and laid out flat over a mica mesh by washing the solution over its surface. Finally, a process used to observe DNA macromolecules known as atomic force microscopy is used to visualize the tiles (see Figure 3.2).

3.3 Tile Assembly Model for DNA Computing

Subsequently, other researchers [2] found additional models of DNA computing. Rothemund and Winfree [13] are credited with defining the tile assembly model for square tiles. Their work was based on a much earlier concept originally known as

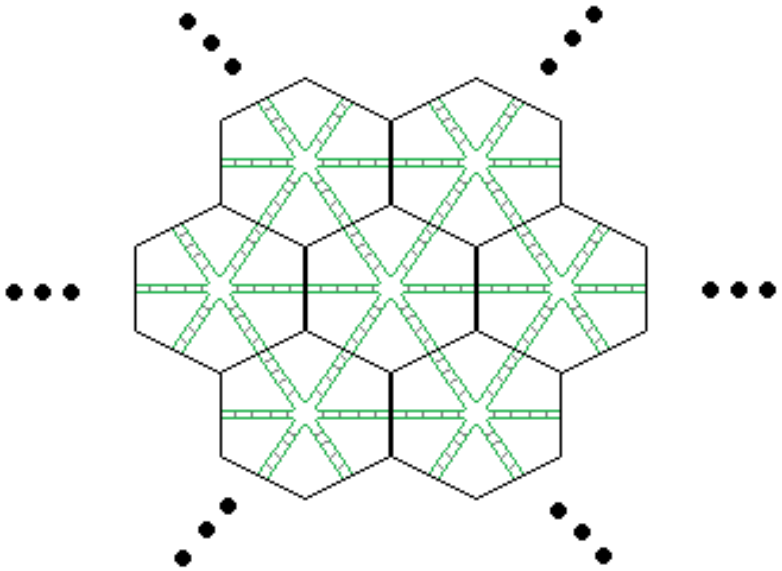

Figure 3.3: The Holliday junction modified to create a hexagon shaped tile.

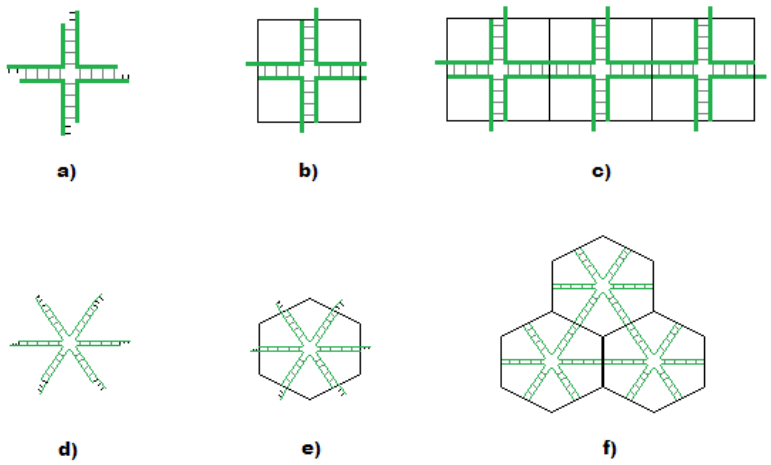

Figure 3.4: 4-sided and 6-sided DNA tiles using Holliday Junctions.

Wang tiles. Wang [15] defined Wang tiles in 1961 to study combinatorial properties of geometric shapes and pattern recognition, but they have all the same properties as the tile assembly model and are therefore useful for computing.

Rothemund [12] was able to design square shaped tiles using four lengths of single stranded DNA combined by a technique known as the Holliday junction. In particular a square tile conssist of a Holliday junction with four single strands. The four single strands represents the four edges of the tile assembly model tile, North, East, South, and West.

Tiles of arbitrary number of sides are possible by extending the Holliday junction to join more single strands of DNA. Figure 3.4 illustrates the molecular bonds between nucleotides that allow the DNA to take on geometric shape and join with other DNA tiles of same shape. In Figure 3.4, (a) and (d) show the Holliday junction for 4- and 6-sided DNA tiles respectively. Figures (b) and (e) show the tile shape overlayed on the DNA structure, therefore demonstrating the connection between the DNA Holliday junction and the geometric shape. Finally, Figures (c) and (f) confirm how the DNA tiles attach to other similarly shaped DNA tiles.

Figure 3.3 highlights the DNA tiles used in the 6-TAM model. Each tile is composed of six single strands of DNA and they are combined around a centre junction to form a six-sided star. Here, the green lines represent the single strands of DNA, the small parallel black lines denote the chemical bonds between complementary nucleotides, and the dark hexagon-shaped outlines are to illustrate how the six-sided Holliday junction can be compared to hexagon-shaped tilings on the 2D plane.

Note that non-regular tilings of the plane are also possible using the DNA tiles

that Rothemund designed. Non-regular shapes are possible as well as combination of polygons. Also the DNA can be made to assemble into 3D objects using the same techniques. These properties of DNA tiles certainly could be considered for future research however the scope of this thesis is focused on square and hexagon regular tilings only. Also it is interesting to note that the tile assembly model could be adapted to tilings on arbitrary surfaces and dimensions, however biological and physical limitations may make it impossible for a physical implementation of such systems but the theoretical definitions and constraints does not preclude non-planar and non-regular tilings.

Rothemund and Winfree make use of these square DNA tiles, and the properties they share with Wang tiles, to define an L-shaped starting configuration of width n and height m and set $\tau = 2$. Figure 2.3 illustrates the concept of a starting configuration. The starting configuration is simply a configuration that is assembled before any of the other tiles in the system are allowed to self-assemble. The starting configuration represents the inputs to the function that will be computed by the self-assembly process. Configuring the starting tiles in an L-shape allows for two inputs to be encoded: one horizontally and one vertically. For example, if the inputs are to be represented as integers, then the starting tiles can represent bits read top-to-bottom and left-to-right. Because $\tau = 2$, the tiles will assemble neatly in rows and columns. This is because the tiles may attach to the configuration at any location where there is already two or more adjacent tiles. The assembly will have a final state because, due to the temperature function constraint, no tiles will be able to assemble outside of the $n \times m$ rectangle given by the starting configuration. The rigorous geometric

property of the final configuration, in particular having deterministic size, is a useful property for defining algorithms in this model. Therefore various algorithms have been implemented using the tile assembly model with square tiles and many variations stemming from the model have arisen as well. Brun [3] uses the tile assembly model for computing addition and multiplication of two numbers using an L-shaped starting configuration with $\tau = 2$.

3.4 Basic Operations with Tile Assembly Model

Brun's solution illustrates an important concept for designing algorithms using the Tile Assembly Model. Namely, his research gives multiple solutions for performing addition and then, selecting one of such solutions for its property to be expanded into larger assemblies, he applies it "iteratively" to obtain the multiplication algorithm. The key factor to observe is that Brun's algorithm for multiplication obtains partial solutions in the form of subsequent additions, not unlike the common "hand written" algorithm for performing long multiplication. Figure C.1 provides a reminder of the long multiplication concept. This is a common theme amongst Tile Assembly Model algorithms, because the regular shape of square tiles, assembled within an L-shaped starting configuration, form to make larger, rectangular super tiles, a concept first introduced by Kari [9]. These super tiles can then be assembled under the same constraints but applied en masse. This design principle allows for algorithm designers to apply a "define once, apply multiple times" approach to the steps in their algorithms. Metaphorically, the process is analogous to functions or subroutines in a modern, high-level computer language.

In more detail, Brun defines his most useful addition system in terms of an XOR operation and a bit shift operation. The bit shift is used to align data such that the bits of the inputs are propagated towards the main diagonal of the configuration. The tiles are defined such that those on the main diagonal will XOR their South and East inputs and pass those value North and West, to represent the sum and carry, respectively. If a carry bit is present, another XOR tile will fit on the West of the main diagonal thus modifying the summation that will occur on the next significant bit. Since the region bounded by the L-shaped starting configuration is defined as square, the main diagonal consistently represents the bitwise application of the sum and carry operations while the other surrounding tiles are passing their input to their neighbours, towards the main diagonal or towards the top row if they are above the main diagonal.

For multiplication, Brun adapts the previous addition method by combining bit shift and addition parts and defining the tiles such that the two operations are applied bitwise from order of least significant to most significant bit. Each row of the final configuration represents the multiplication of one bit, in the "hand written" sense.

Zhang et al. [20] perform long division, but define their tiles such that they assemble in a snaking zigzag pattern. They accomplish their pattern by using tiles which have edges that count as two when being considered in the temperature function. Subsequently, Brun [4, 5] takes advantage of the massively-parallel nature of DNA self-assemblies to compute the solution to various NP-complete problems including Subset Sum and SAT. We will go into detail on Brun's solutions below. A similar problem is attempted by Wang et al. [16]. Their solution is interesting for a number

of reasons that will be explored fully in chapter 5.

3.5 NP-Complete Problems with Tile Assembly Model

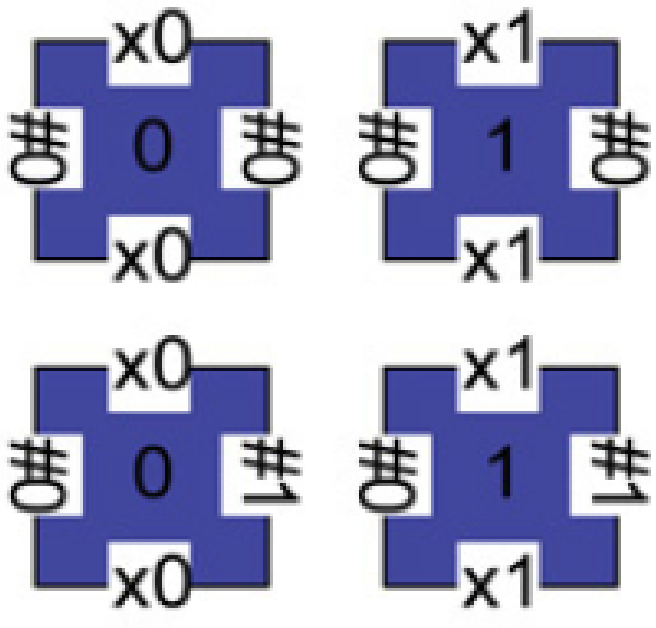

Figure 3.5: Brun [4] illustrates a 4-TAM tile set for copying values from South-to-North and East-to-West.

Intuitively, using non-deterministic assemblies that yield multiple final configurations is the key to solving problems of higher complexity. These tile sets are defined such that a "yes" tile will be present in some final configurations if and only if the inputs represented a "yes" instance of the given decision problem. Otherwise, the instance is solved as "no".

First, we will look at the techniques Brun uses for solving Subset Sum. Brun's solution for Subset Sum extends the same principles used by his arithmetic tile systems, above, combined with this new property of using non-deterministic final configurations to take advantage of the massively-parallel property of DNA computing. Garey and Johnson[7] define the Subset Sum problem as an intractable, NP-Complete problem. Subset Sum requires as input a set of integers S, and an integer target sum T. The goal is to determine whether there exists a subset S' of S such that the sum $\sum_{s \in S'} s = T$. As mentioned, the Subset Sum problem is NP-Complete and therefore is in NP. This implies the existence of a polynomial time algorithm to verify the correctness of a non-deterministically chosen candidate solution. As such, a non-deterministic algorithm for solving Subset Sum can be defined as follows.

Brun uses this algorithm, defined as a Square Tile Assembly Model System with an L-shaped starting configuration and $\tau = 2$, to solve Subset Sum. His solution takes advantage of the technique he used in earlier research to define arithmetic tiles. This time, he chooses to implement Subset Sum by subtracting from a value, instead of adding to the value. The reason will become clear momentarily. For now, consider that the system of tiles that define subtraction is similar to the system of tiles for addition. The bits are passed along the rows and columns towards the main diagonal, where the value is calculated, with carries being passed accordingly. The final output is propagated towards the top of the system. When applied as a super tile, we can make the convention that the output of the super tile that subtracts two numbers is read as the output along the entire North side. Brun also defines a system of tiles that copy a value, from its South side to its North side and accomplishes this

Algorithm 1 Solution for Subset Sum

Input: S set of integers. T target integer.

Output: "Yes" or "No"

for all $s \in S$ **do**

Choose one of the following randomly:

- s is in S'; or
- s is not in S'.

if s is in S' **then**

$T = T - s$

end if

end for

if $T = 0$ **then**

Return "Yes"

else if $T > 0$ **then**

Return "No"

end if

task for the entire set of N rows at a time, for N digit numbers. Therefore when a square section of tiles perform the copy operation, the net result is that instead of a subtraction occurring, the original value is left unchanged (See Figure 3.5). Brun's next step takes advantage of the non-deterministic process for assembling tiles in the Tile Assembly Model. His previous systems for arithmetic and copying tiles were deterministic systems. That is, for some given inputs encoded as a starting configuration, the final configuration produced by the tile system is unique. Using

the non-determinism to his gain, Brun defines specific tiles such that at key positions in the configuration, the tile that may assemble in that position can be one of two tiles, chosen non-deterministically. Intuitively, we say the non-deterministic tile is choosing whether the integer $s \in S$ will be in S', or not. This characteristic is by consequence of the difference of binding domains between the two non-deterministic tiles and how they are interchangeably able to fit neighbouring the starting configuration tiles. Brun defines the starting configuration as having the target value at the bottom, and the integers from S all along the right column. Each integer in S is separated by a tile which is defined such that one of the two non-deterministic tiles can assemble in the first position of the corresponding row. The two non-deterministic tiles differ in that they create a row that initializes which operation (Copy or Subtraction) will occur for the $N \times N$ super tile orthogonal to T and $s \in S$. Up to now, as defined the tile system will create a final configuration whose North side reads a number less than or equal to T. Most notably, the number may be recognized as having zeros in all bits. The system is required to "check" the value of the top row by including one final subsystem of tiles. The tiles are deterministic and will attach in a single row and column along the top and left of the configuration, respectively. The tiles are defined such that if the sum of the chosen values in S' totals less than or greater than T, or in other words if the top row of the final configuration is non-zero, then a final checkmark tile will be allowed to assemble in the top-left most position of the configuration. Otherwise, no final checkmark tile will be present in the final configuration.

Due to the massively-parallel assembly, there will be many configurations produced by execution of the algorithm. The checkmark tile has the property that being

present in a final configuration allows it to be easily detected amongst all produced configurations. Therefore, given a set of possible non-deterministic choices for input, if a non-deterministic choice exists that satisfies the constraints of the problem, it will be detected, in which case we can say the algorithm concluded with a "Yes" output. Otherwise, if the set of all solutions produced provide none with a checkmark tile present, then the algorithm has given a "No" output. Similar to Adleman's solution for Hamiltonian Path, the entire Subset Sum algorithm is defined in a polynomial number of steps, but performs a combinatorially large amount of parallel trials. Recall from Figure 3.2, the amount of biological trials is massive and requires atomic force microscopy to visualize. Brun gave an alternative method for detecting "Yes" outputs to massively parallel systems. The checkmark tile is made with DNA combined with a small amount of microscopic magnetic tracer physically attached to the DNA strands that make the checkmark tile. Therefore, detecting the existince of the checkmark tile, becomes a simple matter of testing the existence of a magnetic force using the appropriate laboratory equipment.

3.6 TAM Comparison with Other Geometric Self-Assembling and Autonomous Systems

The tile assembly model is similar in some ways to well-known research in the area of cellular automata. In fact, Wang tiles are an interesting topic of research in the area of cellular automata and therefore common links exist between the research of cellular automata and the tile assembly model. Kari [8] provides an excellent

comparison and survey on these two models of computation. However there are notable, key differences between cellular automata and the tile assembly model that make their studies independent. Wang tiles are a simplified subset of the tile assembly model, with less constraints. Cellular automata are homogeneous systems of cells with defined properties for states of cells, neighbourhood function for cells, and update rules for cells given their neighbours. Cellular automata are similarly related to Wang tiles. All cells in a cellular automaton update synchronously at discrete time steps. Wang tiles add new tiles to the current configuration in a non-deterministic order.

A specific example of a cellular automaton is the famous Game of Life (GOL) automaton defined by Conway [6]. GOL cells have two states: life and death. The topology of cells is defined over a 2D-Grid and has the neighbourhood of the eight surrounding adjacent cells. Each cell is updated by the following rules:

- A cell with exactly two living neighbours will always maintain the same state in the next time step;
- A cell with exactly three living neighbours will always be living in the next time step; and
- A cell with less than two or more than three living neighbours will always be dead in the next time step.

The connection between cellular automata and Wang tiles is made by a simulation of tile assembly with a cellular automaton. Since both cellular automata and Wang tiles are coordinatized by a grid, that property is analogous. Where it differs is in the

definition of cells or tiles. Wang tiles can be described in terms of cells in a cellular automaton. A cell can be one of some states. Consider a set of states where each state represents a specific Wang tile. Then also include a tile known as the "blank tile". The neighbourhood function for a Wang tile returns the four orthogonally adjacent tiles, that is, the North, East, West, and South tiles. The update rule is implied by the legal set of all tiles that can be placed at a given location and only blank tiles can change from one state to another. Non-blank tiles will never change state.

Although it's true that the tile assembly model can be defined in terms of the cellular automaton model, by examining their actual definitions one can observe some fundamental differences. Cellular automaton cells are analogous to tile assembly model tiles. The cells are homogeneously defined by a set of possible states. The cell can be in one state at a time. In the tile assembly model, tiles come in several variations. Cellular automata have all cells update synchronously at each time step. Tile assembly model tiles will not change once added to a configuration. The purpose of cellular automata are to define a starting configuration and observe the subsequent changes in state. The tile assembly model also defines a starting configuration and assembles new tiles as the computations progress, thus producing output to a computation. Both models can compute functions and are proven computationally universal. One of the key differences, however, is that the tile assembly model is conveniently implemented using natural biological systems, where as cellular automata are generally defined to simulate biological behaviour. Biomolecular computing researchers, such as Adleman [1], can devise and implement DNA self-assembly systems using the tools available to DNA researchers in a biology laboratory.

Chapter 4

0-1 Knapsack Problem

4.1 Problem Statement

The 0-1 Knapsack problem is a well-known NP-complete problem. We are given a capacity C, a target value T, and vectors $\boldsymbol{w}$, and $\boldsymbol{v}$ of size n. The decision variation of the 0-1 Knapsack problem asks if it is possible to assign the values 0 or 1 to elements of a vector $\boldsymbol{x}$ of length n such that the following constraints hold:

- $\sum_{i=0}^{n} \boldsymbol{x}_i \boldsymbol{w}_i \leq C$,
- $\sum_{i=0}^{n} \boldsymbol{x}_i \boldsymbol{v}_i \geq T$.

Intuitively, the problem asks is it possible to choose n or less items from a list, where each item is assigned a weight and value, such that, after totalling the chosen weights and values, the total value exceeds a minimum target, T, and the total weight is below a maximum target, C.

The optimization variation of the 0-1 Knapsack problem is similar. We are not given a value T and instead asked to find T under the following constraints:

Let X be the set of all possible $\boldsymbol{x}$ in $\{0,1\}^n$

- $T = \max_{\boldsymbol{x} \in X} \sum_{i=0}^{n} \boldsymbol{x}_i \boldsymbol{v}_i$
- such that $\sum_{i=0}^{n} \boldsymbol{x}_i \boldsymbol{w}_i \leq C$.

Therefore, the problem is similar to its decision variation, except we are now seeking to maximize value while still enforcing a constraint on capacity. For more information on the 0-1 Knapsack problem, please see Garey and Johnson [7].

4.2 Solution for Square Tiles

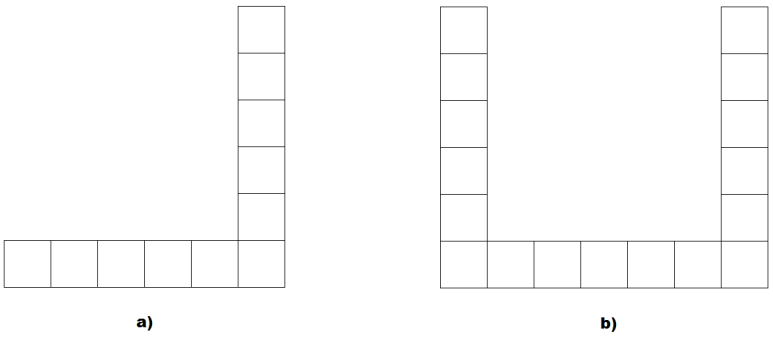

Figure 4.1: (a) An abstracted L-Shaped starting configuration, with labels omitted, using square tiles. (b) An abstracted U-Shaped starting configuration, with labels omitted, using square tiles.

Wang et al. [16] provide a potential solution to the 0-1 Knapsack optimization problem. Their solution is constructed with square tiles. They form a starting con-

figuration that is a U-shaped as opposed to an L-shape (See Figure 4.1). The U-shaped starting configurations are geometrically symmetrical pairs of two L-shaped starting configurations where the bottom tip of the two Ls are joined at the ends. The bottom row of the U-shape encodes the number zero on both the left side and the right side of the centre. This represents that the initial used capacity and value in the knapsack is zero. A tile system for binary addition is defined and mirrored for the left and right sides of the configuration. Similarly, a system to copy the current value is defined for the left and right sides. Refer to appendix A for the definition of Wang et al.'s tile assembly system. The addition system can be interpreted as the current item being selected to be in the knapsack. Likewise, the copy system can be interpreted as the current item not being put into the knapsack. The selection process occurs non-deterministically through use of the fact that where more than one tile may legally assemble at one position, and the tile used for that production will be non-deterministically selected by the biochemical process which facilitates the production. The left and right sides of the assembly are separate and joined by a column of tiles from a separate system known as the "verification" tiles. This latter system acts as a verification step designed to coordinate the choosing of variables in the left and right side, such that the weight and value of individual items being added or left out of the knapsack are consistent with each other. In particular, the middle column verification tiles are defined so that if the left side is performing an addition function, then the right side must also be performing an addition function, otherwise the final configuration will contain empty spaces in this column. It also holds true for the case where both sides are performing copy operations. The middle column

cannot assemble completely if at any point an x_i was chosen on one side and not on the other side. x_i must be chosen consistently on both sides in order for the final configuration to assemble completely.

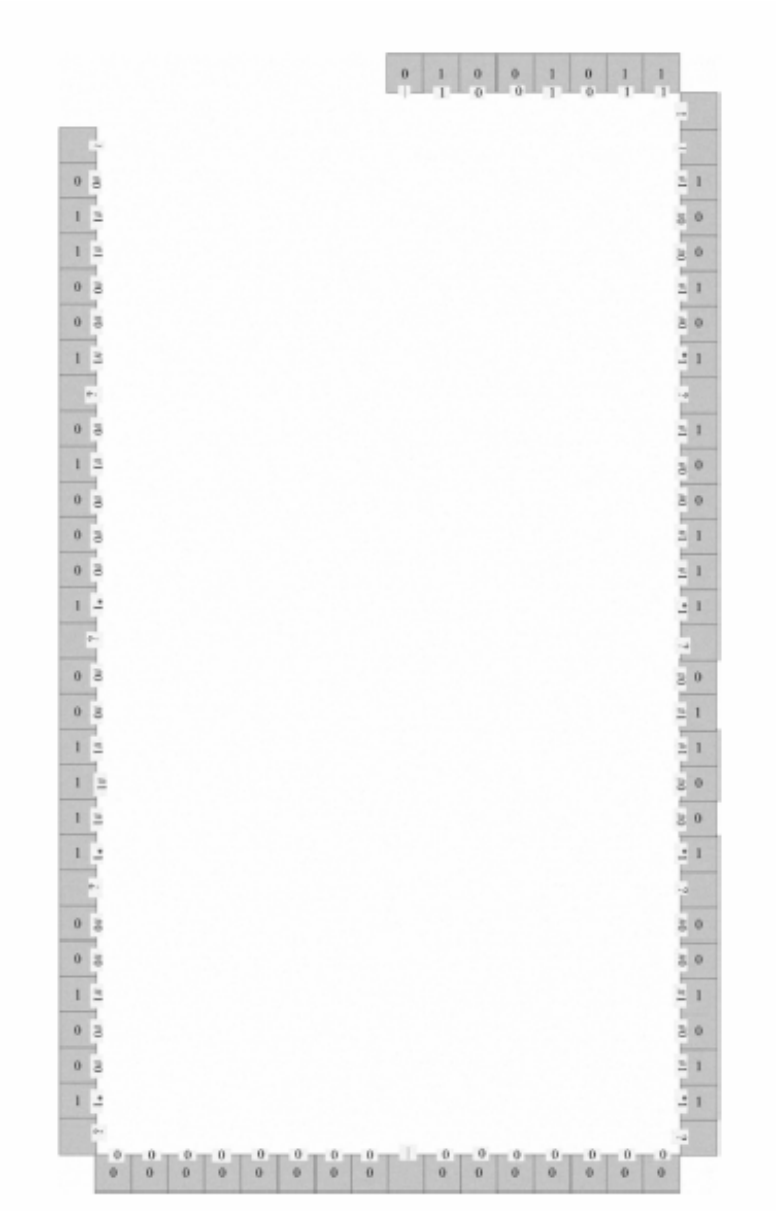

Figure 4.2: A sample starting configuration using the tile set defined by Wang et al. [16]. Notice the starting configuration is mostly U-shaped however there is an additional eight tiles above and to the right of the "U".

As the assembly reaches the top of the U-shape, there is one more portion of

starting configuration tiles that exist as a partial top to the rectangular shape (See Figure 4.2). These tiles encode the capacity and provide a final check to the correctness of the current solution. Recall that the optimization problem requires that the total capacity of the chosen objects be at most C. Therefore, another verification tile set is defined in the partial row between the totalled capacity and the target capacity. This tile set assembles in that space if and only if the totalled capacity is less than or equal to the target capacity. When this final verification step has successfully assembled, the checkmark tile is then allowed to attach at the final available position.

Wang et al. [16] describe that finding the optimal solution requires observing all successful assemblies, quoting the reporter strand method as being the implementation of choice to this end. Katz [10] describes the reporter strand as an additional DNA macromolecular structure that assembles within a self-assembly, physically attached to the tiles. The elements of the reporter strand represent a vector of data whose elements have one-to-one mapping with the labels of the tiles they are attached to. The strand can then be isolated from a final configuration and the values extracted through various biomolecular operations, such that a final configuration maps to a final value. Therefore, the optimal solution of a set of final configurations representing candidate solutions of a 0-1 Knapsack instance is the one that yields the highest totalled value. Unfortunately, the method of Wang et al. for solving the 0-1 Knapsack optimization problem with square tiles is flawed.

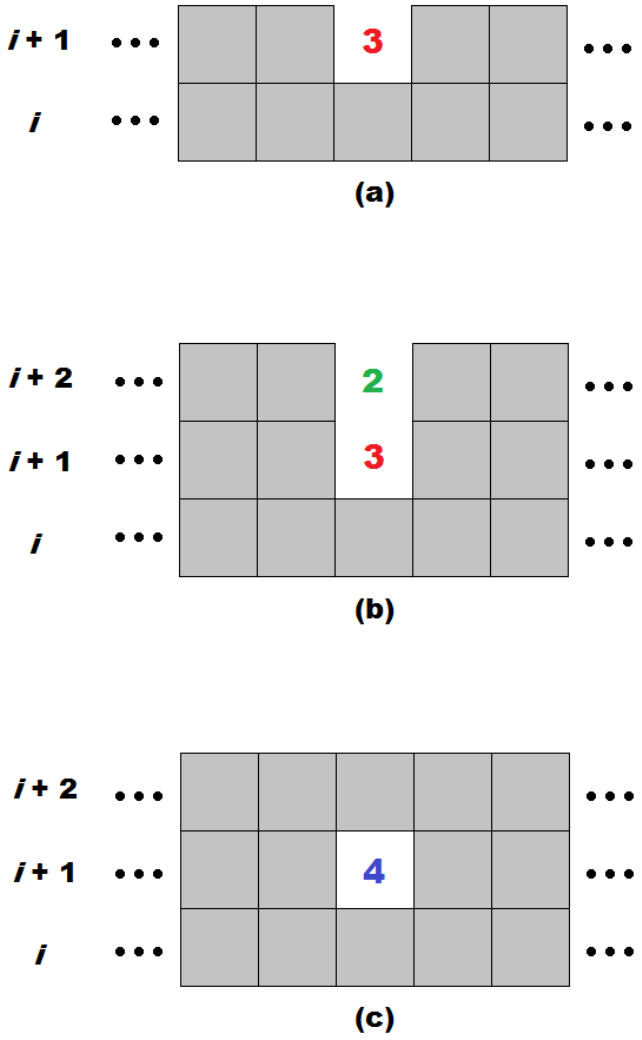

Figure 4.3: The fundamental flaw in U-shaped 4-TAM assemblies is the possibility of holes assembling.

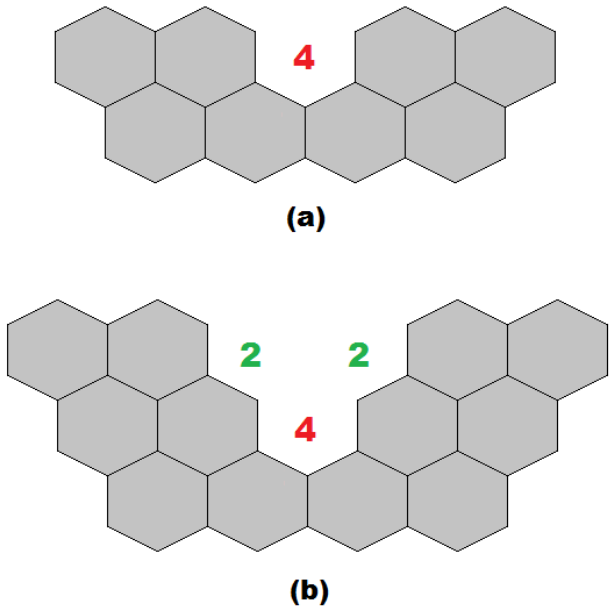

Figure 4.4: Compare Figure (a) and (b) with the respectively named illustrations of Figure 4.3.

Algorithmic computations provide the easiest means to demonstrate the advantage of hexagonal tiles used in self-assembly. Consider the comparison given by Figures 4.3 and 4.4. Figure 4.3 gives a step-by-step partial assembly of a possible scenario that may invalidate use of the 4-TAM system for these shape of assemblies. The steps are described as follows:

a. Row $i+1$ has assembled save one position $(x, i+1)$. This position has three neighbours, therefore the temperature function $\tau = 2$ allows for a tile to be placed here.

b. After the assembly continues from (a) for some time, row $i+2$ has assembled save one position $(x, i+2)$. The temperature function $\tau = 2$ allows a tile to be

placed in position $(x, i+2)$ since there are currently two adjacent neighbours. This position is directly above the empty position in the row $i+1$ below it. At this point, there is nothing to stop either $(x, i+1)$ or $(x, i+2)$ from being filled with an appropriate tile, if the edges match.

c. The assembly continues from (b) for some time. The last empty position in row $i+2$ is filled with a tile before the last empty position in row $i+1$. Position $(x, i+1)$ now has four neighbours and the temperature function $\tau = 2$ would still allow a tile to assemble at this position. However, it's possible that no tile fits this position. Furthermore, note that for any tile that would fit in this position, it will not affect the outcome of any other tiles because all of its neighbours have already been chosen. These two facts together imply that were the system designed such that only complete rows imply a true positive, then there is no such guarantee while rows may be fully completed above adjacent rows with holes.

   Compare this situation to Figure 4.4 where the lack of tile in row i guarantees that no tile will be able to fit in all positions of row $i+1$. This is illustrated by (b) where the empty space with 4 neighbours must be assembled before either the North-West or North-East positions can have 3 or more neighbours. Otherwise, since $\tau = 3$, no tile will be able to assemble in the North-West or North-East neighbouring positions of the empty space in row i.

Figure 4.5 illustrates the striated way data can flow through a tile assembly. First, refer to (a). Square Tiles are assigned input and output edges. Since there are four edges to a square, we have two pairs of inputs/outputs: the North-South pair and the

West-East pair. We denote the different pairs with different colours, red and green respectively. Contrariwise, hexagons have three pairs of symmetrically opposing edges available. Part (b) shows we have the trivial transformation (transliteration) of square tiles to hexagons by simply mapping two input/output pairs and leaving the third pair unused. Technically, we use a "dummy" value that carries no additional meaning in the current system, shown as the white arrows in part (c). Without loss of generality, we denote the "dummy" edge in figures as being an edge with no value specified.

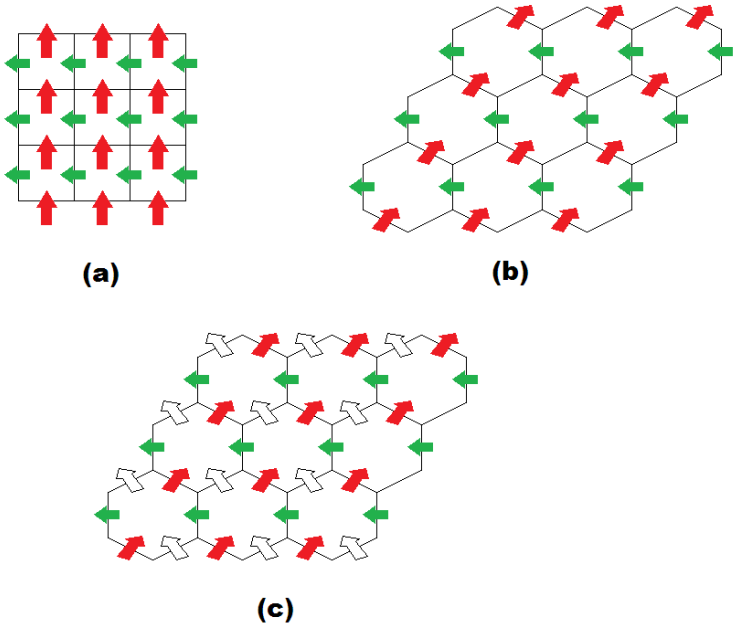

Figure 4.5: Comparison of edge-wise transmission of data for 4-TAM and 6-TAM tiles.

Every tile assembly system is defined with the parameter τ, used to set a lower limit on the required total binding strength of all adjacent bonds when a tile is added to the configuration. For L-shaped starting configurations, the value of τ is

2, and Brun [3] proves that the combination of L-shaped starting configuration and $\tau = 2$ guarantees that any row i in the configuration must have a tile attached in every position before the $i + 1^{\text{th}}$ row can have the same. Similarly, every column i must have a tile attached in every position before the $i + 1^{\text{th}}$ column is filled. These facts combined imply that the tile in the n^{th} row and n^{th} column is the last tile to be assembled if and only if every other space in the configuration has already been filled. Brun [3] as well as Wang et al. define their tile systems under that assumption. However, the problem lies within the fact that Wang et al. do not give a proof that a U-shaped configuration also guarantees this assumption. In fact, they also do not provide an explicit value for τ. These setbacks yield a system that gives false positives as well as false negatives. In other words, for all possible values of τ, their system may not work in all cases, due to the shape of their starting configuration. These flaws are stated as a theorem and proven in Chapter 5.

One of the other issues with this square tile solution to the 0-1 Knapsack problem is that it attempts to solve the optimization variation. Due to the fact that NP-Complete problems are intractable, there are an exponential number of candidate solutions to any given instance of the problem. The strength of using the tile assembly model to solve intractable problems is that the massively parallel nature of DNA self-assembly allows the user to generate all solutions in a feasibly more efficient amount of time, compared to the best exhaustive computational methods of modern, electronic hardware. We could say that a costly solution has been made more affordable. However, Wang et al. [16] do not specify details as to what method is used regarding the extraction of the optimal solution from a set of candidate solutions. It

is not clear how they propose to verify if a given solution is optimal, short of examining all possible solutions. It is also unclear if Wang et al. provide a method better than exhaustive search of a exponential search space with respects to the input size, which is not a feasible answer given the tools to accomplish the task.

4.3 Hexagon Tile Solution For 0-1 Knapsack

All is not lost for solving the 0-1 Knapsack problem. Wang et al. [16] were correct to assume that a U-shaped starting configuration could solve the problem with tile sets, however square tiles are not apt for the job. Hexagon shaped tiles are however quite capable of using U-shaped starting configurations to provide tile assembly solutions to NP-Complete problems. In particular, due to the change in shape of squares to hexagons, we are allowed to use starting configurations that are bounded on the bottom and on both sides. The concept of rows remains analogous, however the axis previously representing columns is now analogous to two overlapping directions. Given the direct transliteration from a square system into a hexagon system, the majority of the hexagons do not require "two columns" and therefore leave the edges in one direction unlabelled (see Figure 4.5). As such, we define a hexagon tile U-shaped starting configuration such that the top-left tile is the last to assemble and it assembles only if each previous row has been completely filled, meaning a tile has attached in every position of each row. The other improvement my hexagon solution will have over the square solution is that it will solve the 0-1 Knapsack decision problem instead of the optimization problem. Thus, it is still an NP-Complete problem, however it will have two restrictions, namely the capacity and the target value. The

correct solution will have a totalled capacity of chosen objects less than the knapsack's capacity and totalled value greater than the target value. These values will be calculated through addition of each value (or capacity) added to the negative of the target value (or capacity) encoded in two's complement binary notation. Then along the top row we apply verification tiles to check that we have not exceeded our maximum and have reached our minimum. We are able to check if the sums exceed a target if the most significant bit (MSB) is zero since we started off with a negative number (by definition of two's complement binary notation, the MSB of a negative number is one). A similar check is applied to the capacity, this time verifying the MSB is one, implying that the sum of capacities for each item chosen does not exceed the maximum capacity. If these constraints hold, then the system allows a checkmark tile to be placed in the final position thus producing a final configuration where a tile has been attached in each position.

Chapter 5

A problem with square tiles for 0-1 Knapsack

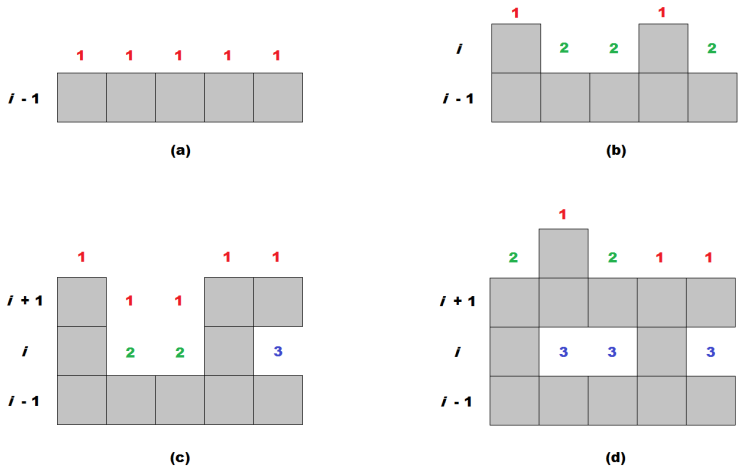

Figure 5.1: When $\tau = 1$, tiles may assemble adjacent to any other tile. This may cause the appearance of holes.

Intuitively, we want to show that for any possible value of τ, the U-shaped starting configuration of a square tile system may yield a final configuration containing

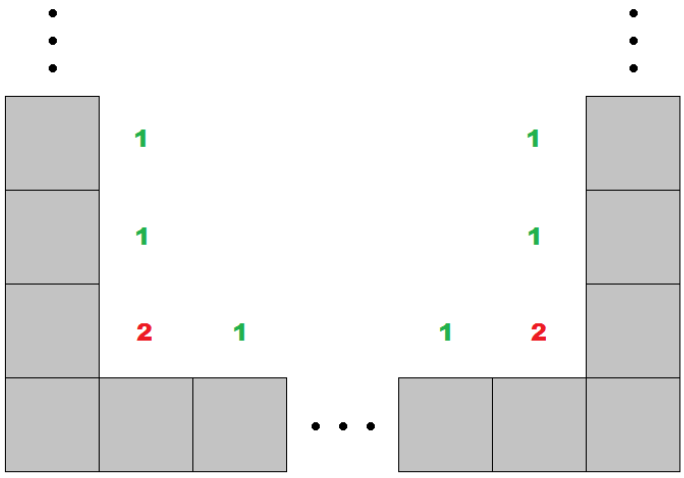

Figure 5.2: U-shaped starting configuration with square tiles. When $\tau = 4$, no tiles will be able to attach to such a system.

both the checkmark tile and a hole. We express the concept of a "hole" by saying there exists a position in some configuration with neighbouring edges on all sides totalling greater than or equal to τ, however no defined tile exists that will fit in this position. Consider Figure 5.2 which illustrates an arbitrary U-shaped starting configuration with square tiles. The numbers in empty positions provide the total number of adjacent neighbours that are not empty, using formula 2.6. Note that the greatest number in the figure has the value 2. Therefore, if the system was defined with this starting configuration, and $\tau = 4$, there would be no available positions for additional tiles to assemble. This is the trivial case where a starting configuration is also the unique final configuration for any and all tiles that could be defined.

Theorem 1. *For all τ with $1 \leq \tau \leq 4$, there may exist a U-shaped starting configuration A and position $p = (x, y)$ where the following statements are true:*

- $A \xrightarrow{*} B$,
- B *is a final configuration,*
- $B(p) = \text{EMPTY}$,
- $\sum_{d \in \boldsymbol{D}_4} \max_{\sigma \in \Sigma} g(bd_{d^{-1}}(B(\text{Adj}_d(p))), \sigma) \geq \tau$, *and*
- *the checkmark tile exists in* B.

Proof. The Wang et al. System operates under the assumption that a hole will always imply the candidate solution is "no". This is a common characteristic of the systems we are considering. First, we will define this "design technique" for TAM.

When solving NP-Complete problems using the Tile Assembly Model, the inputs are encoded on the bottom and side(s), then the tile systems non-deterministically determine a candidate solution. At some point in the assembly, verification tiles will assemble if and only if the candidate solution is positive and such that the verification tiles will complete the assembly. Lastly, the checkmark tile will assemble if and only if a complete set of verification tiles have assembled. In other words, the tiles must be defined such that:

- checkmark $\Leftrightarrow$ verification was a success,
- verification was a success $\Leftrightarrow$ candidate solution is positive, and
- candidate solution is positive $\Rightarrow$ problem instance "yes"

Note: $F \Leftrightarrow F \Leftrightarrow F \Rightarrow T$ is logically well-formed where $T \Leftrightarrow T \Leftrightarrow T \Rightarrow F$ is not.

Therefore we are requiring that a system which attempts to follow these rules must have the checkmark tile assemble last and it implies no holes are present, because the verification tiles will not allow it.

Continuing, when $\tau = 1$, or 4, the case is trivial. Figure 5.1 illustrates the case for $\tau = 1$ and demonstrates the squares can assemble adjacent to any neighbour during any step of the assembly. This example illustrates how holes may appear for some generalized sub-section of an assembling TAM with $\tau = 1$ and therefore it shows $\tau = 1$ cannot guarantee the absence of holes. The numbers in EMPTY positions represent the number of adjacent neighbours given by formula 2.6. In each step, (a) through (d), a few more tiles are assembled. The exact order of tile assembly is not necessary information to illustrate this example. The important point to emphasize is that row i may be completed before row j or vice versa. The verification tiles may assemble first, including the checkmark tile, thus bypassing, and therefore not logically following, the assembly of the non-deterministic decision tiles or the other groupings of tile sets that are included in the system. Therefore, $\tau = 1$ cannot guarantee the absence of holes.

With $\tau = 4$, no tiles may assemble. Figure 5.2 shows the fact that the system will always provide the negative answer only.

When $\tau = 2$, or 3, things can be different. Figure 5.3 illustrates that $\tau = 3$ only works when there is a single column to assemble. In that case, holes are not possible, but otherwise it is the same issue as case $\tau = 4$. No tiles may assemble and the answer is always the negative.

Case $\tau = 2$ is the most interesting case and is likely the assumption for the Wang

et al. tile system. The issue arises whenever a row above another row assembles a column before the lower row does, for example, when a situation assembles such as that in Figure 5.4. In this figure, $\tau = 2$, as is common with TAM solutions using square tiles. This example uses a U-shaped starting configuration. In Figure (a), rows y and $y+1$ are assembled except for positions P and Q. These positions are empty. Note that the number of adjacent neighbours for P and Q both exceed the value 2. In (b), the issue is demonstrated. Notice that position P may legally assemble before position Q. Moreover, the rules for TAM make no requirement at all that position Q be filled with a tile. Even if Q did subsequently contain a tile, no other subsequently assembled tiles would depend on which tile assembled at position Q. Let position $P = (x, y+1)$, and $Q = (x, y)$.

For directions $d_1, d_2, d_3, d_4 \in \boldsymbol{D}_4$,

a. $d_1 \in \{E, S, W\} \Rightarrow A(Adj_{d_1}(Q)) \neq \text{EMPTY}$,

b. $d_2 \in \{E, W\} \Rightarrow A(Adj_{d_2}(P)) \neq \text{EMPTY}$,

c. $d_3 \in \{N\} \Rightarrow A(Adj_{d_3}(Q)) = \text{EMPTY}$, and

d. $d_4 \in \{N, S\} \Rightarrow A(Adj_{d_4}(P)) = \text{EMPTY}$.

Therefore, assuming a tile $\boldsymbol{t}_1$ exists in T such that $g(bd_E(\boldsymbol{t}_1), bd_W(A(Adj_W(P)))) = 1$ and $g(bd_W(\boldsymbol{t}_1), bd_E(A(Adj_E(P)))) = 1$, then $\boldsymbol{t}_1$ can assemble at position P. Therefore, the assembly of position P does not depend on the assembly of position Q. If no tile fits in Q, namely $\forall \boldsymbol{t} \in T, \sum_{d \in \boldsymbol{D}_4} g(bd_d(\boldsymbol{t}), bd_{d^{-1}}(A(Adj_{d^{-1}}(Q)))) < 2$, then the assembly should yield the negative answer, but as we can see, a positive result is not

impossible. Extrapolating the assembly, the issue of Q being EMPTY does not preclude A from assembling subsequent tiles, producing a final configuration that may or may not contain a checkmark tile. Therefore, there is a violation of the constraint: checkmark $\rightarrow$ problem instance "yes". We conclude that the TAM solution for 0-1 Knapsack proposed by Wang et al. is invalid for $\tau = 1$, 2, 3, or 4 (all possible values) and is therefore not a valid solution. $\square$

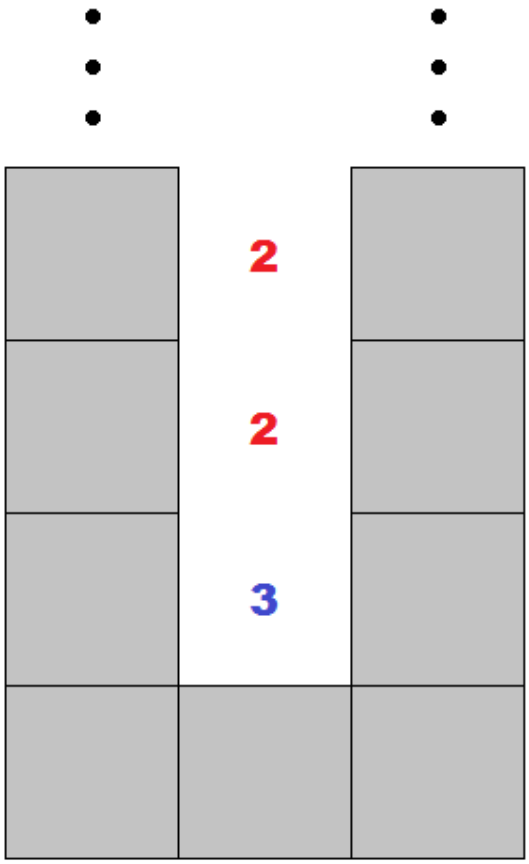

Figure 5.3: When $\tau = 3$, there is one case of the U-shaped starting configuration that allows for tiles to assemble. The arbitrary case follows the example illustrated in Figure 5.2 and by the same logic, that figure depicts a final configuration when $\tau = 3$.

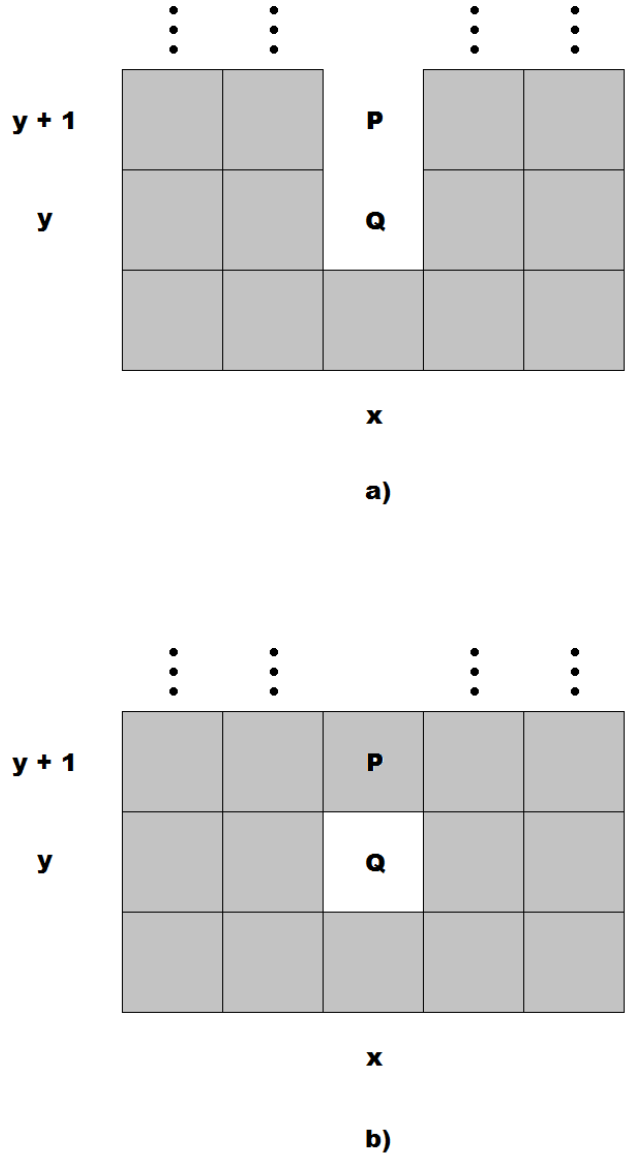

Figure 5.4: When $\tau = 2$, holes may appear at position Q, if a tile attaches at position P.

Chapter 6

Hexagonal Tile Operand Subsets

In the previous chapter, we demonstrated that square TAM solutions fall short at solving the 0-1 Knapsack problem, with a U-shaped starting configuration. In the following sections, we'll provide a satisfactory solution using hexagonal tiles and a U-shaped starting configuration such that the previously described shortcomings are completely resolved. The solution will be built up through a combination of first principles of TAM subsystems. We'll begin by defining complete subset systems to solve simpler sub-problems. We'll define some basic arithmetic functions including the identity function, addition, subtraction, and multiplication. Then we will show how combining these sub-problems, particularly addition and the identity function, can solve the 0-1 Knapsack problem. Finally, we will show proof that the combined set of tile systems completely solves the 0-1 Knapsack problem with hexagonal tiles.

6.1 6-TAM Arithmetic — Copy Tiles

6.1.1 Unidirectional Copy Operation

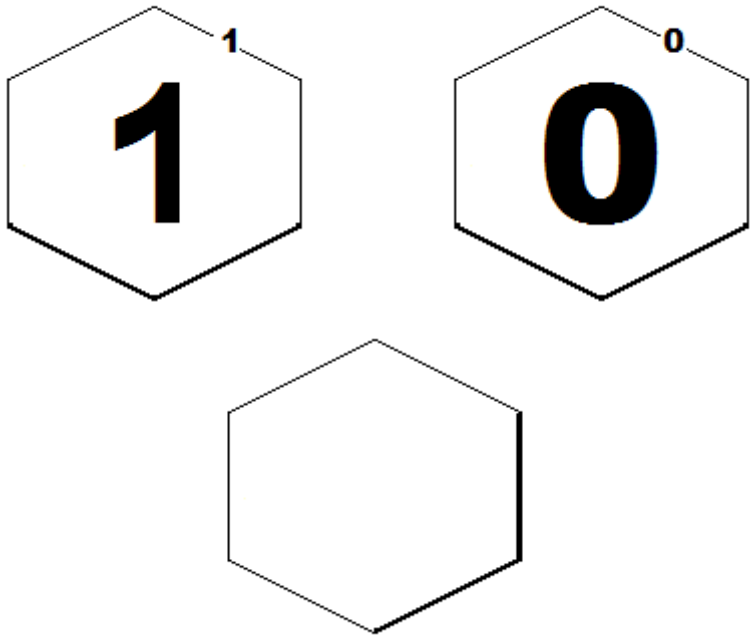

Figure 6.1: Hexagonal Unidirectional System — Start Tiles.

We define two varieties of Copy Operations using hexagonal tile systems, namely, Unidirectional and Bidirectional Copies. Both versions will be adapted for reuse in the other arithmetic operations, below. We begin by presenting Figure 6.1. The Figure defines three starting tiles, with the first two as follows.

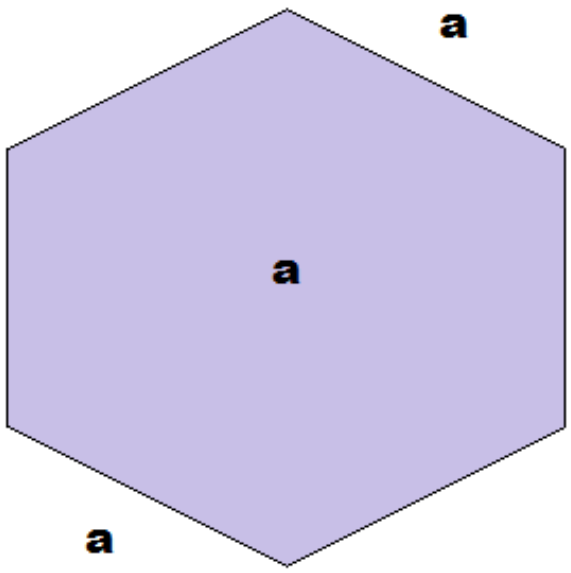

Figure 6.2: Hexagonal Unidirectional System — Unidirectional Copy Tiles.

$$\boldsymbol{t}_a = \langle \text{BLANK}, \text{BLANK}, a, \text{BLANK}, \text{NULL}, \text{NULL} \rangle \text{ and } v(\boldsymbol{t}_a) = a, \forall a \in \{0, 1\}. \tag{6.1}$$

These hexagons have the bits 0 or 1 placed on the NE side as a label. The third tile, the tile without labels, is only used to space the starting configuration and carries no computational meaning.

Next, we define the tiles represented by Figure 6.2. Here, we use the symbol a to mean $a \in \{0, 1\}$. Therefore, taking a over both values in 0 and 1 yields two possible

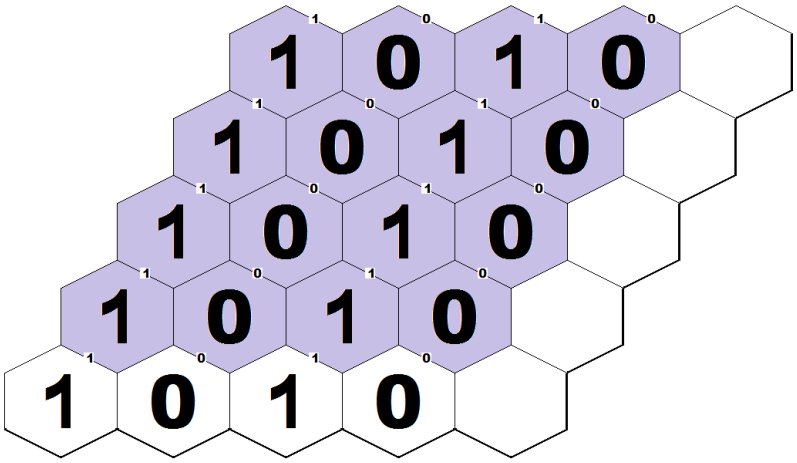

Figure 6.3: Hexagonal Unidirectional Copy Example.

tiles labelled 0 and 1 respectively. Notice the value of the SW edge and the NE edge are both the same, and the label is the same also. Since the temperature function in this system is $\tau = 3$, a tile in this system must attach to 3 or more neighbours. Therefore it is only able to attach to an existing configuration on its SW, SE and E edges. As a result, it is adding its own NE edge to the set of candidate binding domains available for another tile to attach to. We can look at this assembly as tile $\boldsymbol{t}$ at row i passing a value a from row $i-1$, through row i, to row $i+1$. The relative ordering of the bits will never change per row. Hence we refer to this set as performing a copy operation, as it perpetuates the same values unmodified from South to North throughout the subsection of assembly that it covers.

Figure 6.3 illustrates a final configuration produced by the starting configuration encoding the binary number 1010_b along the bottom row. Observe that for each subsequent row above the first, the labels will read 1010. The top row contains tiles whose NE edges are unattached to any other tiles. Therefore, we could have

added more rows to this configuration by simply increasing the height of the starting configuration. Moreover, we could allow another system of tiles to use the output of this configuration as their starting configuration. We will present an example of this technique below, as we introduce a strategy for combining tile subsystems to solve more complex problems.

6.1.2 Bidirectional Copy Operation

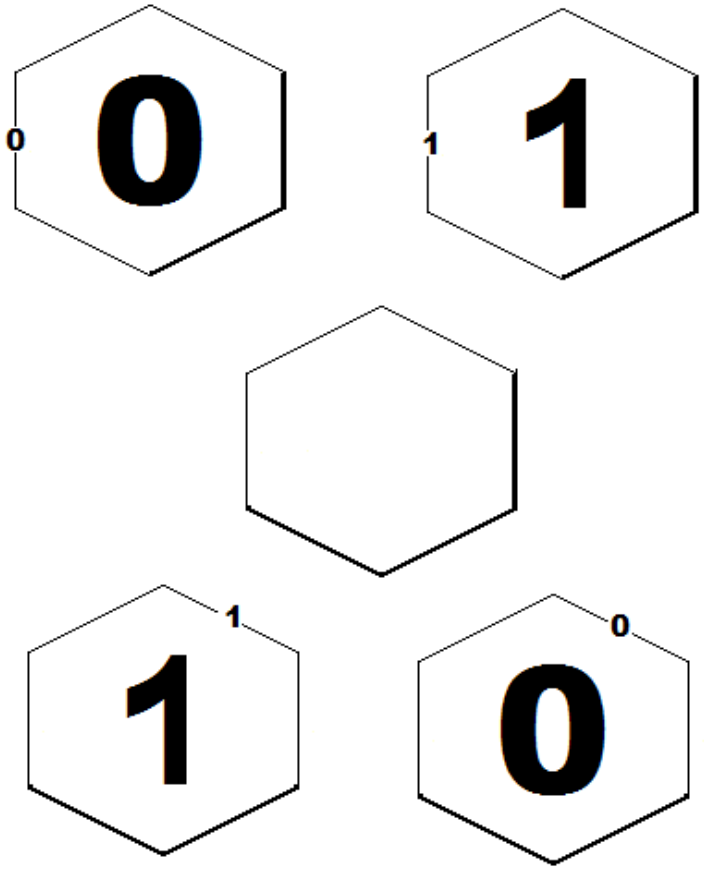

Figure 6.4: Hexagonal Bidirectional Copy System — Start Tiles.

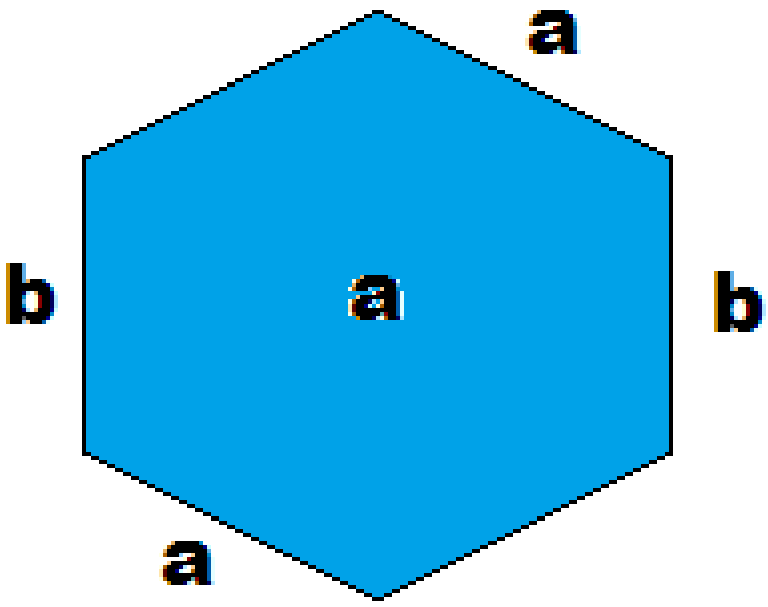

Figure 6.5: Hexagonal Bidirectional Copy System — Bidirectional Copy Tiles.

The tile set we define in 6-TAM is the Bidirectional Copy System. Figure 6.4 defines the five tiles comprising the starting configuration we use in the example depicted in Figure 6.6. The tiles in Figure 6.5 define the remainder of this set. Notice the SW, NE, and label are the same symbol namely, $a \in \{0, 1\}$. This is similar to the Unidirectional tiles from Figure 6.2. Where the Bidirectional tiles differ is their West and East edges. They define a symbol $b \in \{0, 1\}$. The symbol b allows binary values to be copied from the East to West direction. This lateral copy operation works in the same fashion as the NE to SW copying from the Unidirectional system. More specifically, it works at the same time as performing the copy between rows. Since this system copies across rows and columns, the set of binary digits $\{0, 1\}$ yields four permutations. Hence the four definitions of tiles as given by Figure 6.5 are evalu-

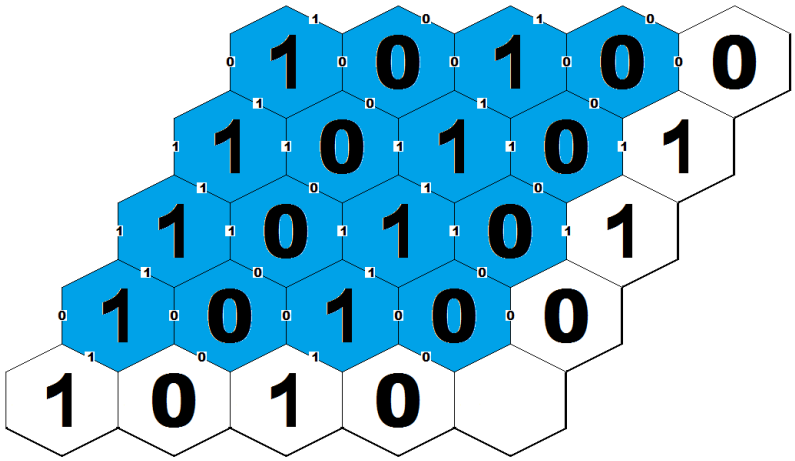

Figure 6.6: Hexagonal Bidirectional Copy Example.

ated from the formula $\boldsymbol{t}_{a,b} = \langle b, \text{BLANK}, a, b, \text{BLANK}, a \rangle$, $v(\boldsymbol{t}_{a,b}) = a$. Geometrically, the Bidirectional tiles serve to move data across subregions of the assembly. When creating larger assemblies comprising of multiple tile sets as this one, it would be algorithmically correct to use either Uni- or Bidirectional tile subsystems. The deciding factor may come down to how and if the tiles fit into the region. As such, it may be convenient to use Bidirectional tiles where Unidirectional tiles would not fit, however would otherwise suffice. We provide this hypothetical example to motivate the utility of the copy subsystems. We will demonstrate more concrete examples in this chapter and the next that combine copy tiles with other systems.

Figure 6.6 shows an example of copying two values in a region. The bottom value in the starting configuration is $1010_b = 10$ and the output on the top row is $1010_b = 10$. Hence the value was copied from the bottom to the top of the subregion. Similarly, observe the value encoded on the right side of the starting configuration. The value $0110_b = 6$ is equivalent to the output along the left edge. Note, the label

of the tiles is not the output, it is the value on the West edge that is being referred to as output. The values on the West edge are $0110_b = 6$ hence the values were copied in the right-to-left direction as well.

6.2 6-TAM Addition Tiles

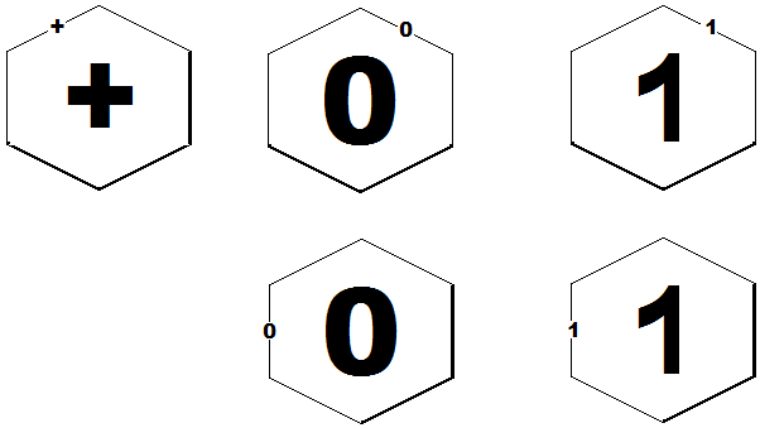

Figure 6.7: Hexagonal Addition System — Starting Tiles.

The next tile system we will define is the addition system, as depicted in Figure 6.10. This set of tiles is arguably more interesting than the simple copy systems, in part because it comprises the tiles of those systems and adds a few more. The addition system provides a 6-TAM solution for calculating the sum of two integers. The inputs are read on the bottom and right edges of the starting configuration. Note, that we use zeros to pad the bits on both the bottom and right inputs. We use two's complement arithmetic therefore with sufficient padding, an overflow error will not occur. Furthermore, the bottom and right inputs are required to be the same

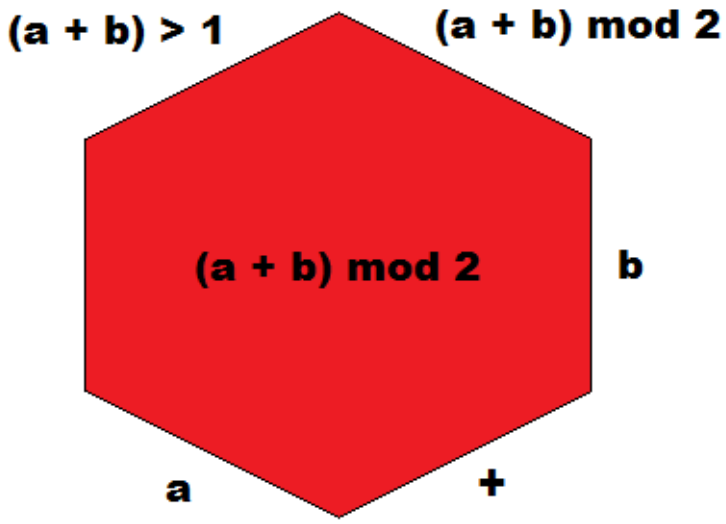

Figure 6.8: Hexagonal Addition System — Deterministic Add Tiles.

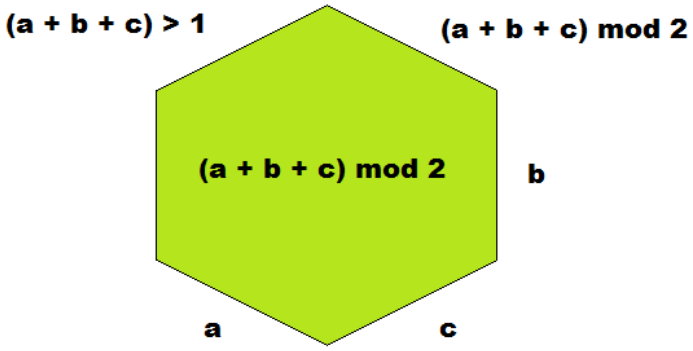

Figure 6.9: Hexagonal Addition System — Addition Tiles.

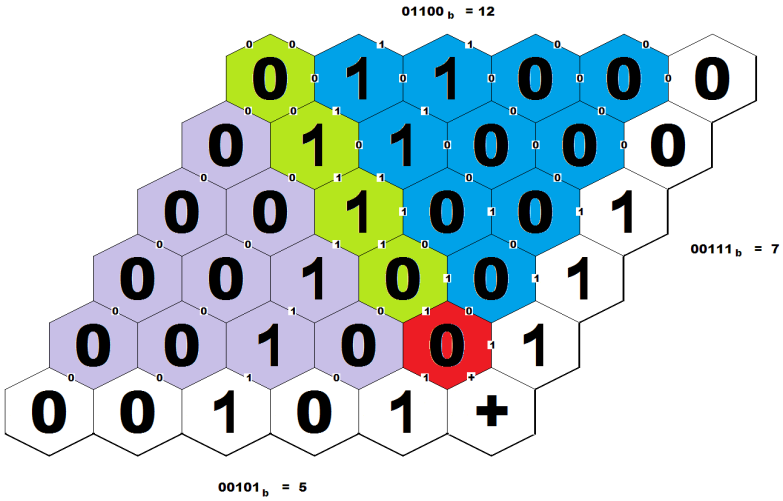

Figure 6.10: Hexagonal Addition Example.

length, because the addition operation occurs along the main diagonal therefore the subregion needs to have equal height and width for all bits to be summed.

The starting configuration, defined in Figure 6.7, is nearly identical to that used for the copy tiles. Notice we use an explicit symbol, the '+' character, to identify this starting configuration as having an explicit purpose, namely the summation function of two integers. The symbol is used both as a label on one of the starting configuration tiles, as well as binding domain on the edge of said tile and the tiles defined in Figure 6.9. Also note that unlike the 1's and 0's we've used until now, the + symbol expresses no value. Instead, it serves to disambiguate the subset of tiles available to legally assemble in some given position whose neighbouring tile contains the symbol on another edge with no neighbour. As previously foreshadowed, we will soon demonstrate an example building upon this one and the previous two will reuse many of the same tiles. Therefore, although it may appear in this example, illustrated by

Figure 6.10, that the symbol is being used as a superfluous label, we are introducing a naming convention that will be required in a subsequent example to assist in readability as well as controlling which subset of some subsystems may assemble in the adjoining subregion.

The next set of tiles, in Figure 6.8, is used as a sort of start tile in the assembly process of the addition system. The high level description of this tile is given the first two bits of the two binary numbers as input, the tile provides the sum and carry for those two bits, as output. These tiles have as consequence of their design that they will be the first tile to assemble after the proper starting tiles have been established and pre-assembled. In this example alone, there is not a reason to require the red tile to be defined separately as an additional tile to the already sufficient green tiles. In later examples, the concept will be relevant so the outcome of the system's result will be dependent on the tile placed first when more than one tile is available for being chosen non-deterministically. We define this tile as $\boldsymbol{t}_{+_1} = \langle \text{BLANK}, (a+b) > 1, a+b \mod 2, b, +, a \rangle$, and $v(\boldsymbol{t}_{+_1}) = a + b \mod 2$, $\forall a, b \in \{0, 1\}$. The formulas on the NW and NE edges are very simple to understand when described in words. The formulas simply output the two relevant values resulting from one step of addition of two binary digits, namely the sum (NE) and the carry (NW). Here, we use > 1 to represent a function that maps integers to $\{0, 1\}$. The function is defined as follows:

$$x > 1 = \begin{cases} 0 & : \text{if } x \leq 1 \\ 1 & : \text{if } x > 1 \end{cases} \tag{6.2}$$

In other words, if the LHS is 2 or more, then the output is 1, otherwise the output is 0. The operation $\mod$ is the modulo operation. Therefore the sum of two bits

maps to {0, 1} consistent to if we dropped the non-MSB bits.

The next tile subset in the addition system is defined in Figure 6.9. Notice that the definition of these tiles is nearly identical to the previous. The only difference is their SE edges. This time, we've chosen the variable c to represent the carry bit, where $c \in \{0, 1\}$. Since there is a third bit as input, the carry, we must include it in the formulas that make up the NW and NE sides and labels. Hence the definition of this tile updates to $\boldsymbol{t}_{+_2} = \langle \text{BLANK}, (a+b+c) > 1, (a+b+c) \mod 2, b, c, a \rangle$ and $v(\boldsymbol{t}_{+_2}) = (a+b+c) \mod 2$, and $\forall a, b, c \in \{0, 1\}$.

Now we will describe the relation and significance of the addition system tiles that we have thus far defined. Figure 6.10 illustrates a complete example of a final configuration that was produced from an L-shaped starting configuration encoding the numbers 7 on the right edge and 5 on the bottom. First, we will describe the high-level effect of the system, then explain the interaction of the lower-level components, making reference to Figure 6.10 throughout. At the high-level, this system computes the sum of the two binary numbers $00101_b + 00111_b = 01100_b$. Notice the labels of the top row read from left-to-right (not including the white starting tile) yield the values 01100. Also, the NE edges of those tiles show the same values since the NE edge is $v(\boldsymbol{t}_{+_2})$ for the tiles in this system. Next, consider the colours of tiles in this assembly. We've purposefully colour-coded the tiles so tiles with like meaning are of the same colour. Hence, all four tile types we've defined as being part of the addition system are represented in this final configuration. Therefore, we can refer to the colours in this example to describe visually what is happening computationally.

Please note, so as not to be misled, that the colours in this example are grouped

contiguously, however this is not a requirement in the general case. We will demonstrate more involved systems where the tiles of similar design may be more spaced apart and not adjacent. The red tile, defined in Figure 6.9 is the first tile to attach to the white starting configuration tiles. The red tile computes the sum and output the carry bit on the NW side and the sum bit on the NE side. Following the diagonal columns of tiles in the repeated NE direction, the label stays the same. This is of course expected because the blue tiles defined in Figure 6.5 are the Bidirectional copy tiles that repeat their value in the NE direction continuously. Now consider the green tiles, defined in Figure 6.9, that extend across contiguous neighbours in the NW direction, starting from the first NW neighbour of the red tile. These tiles perform the subsequent addition of remaining bits of x and y and the carry bit of the previous step of the addition. The bits of the inputs are passed along from the initial position of the starting configuration in the most intuitive way available. The purple region of tiles represent the unidirectional copy tiles from Figure 6.2. Therefore, they pass the bits up to the neighbouring tiles in their NE neighbour, i.e., the row above them. The blue tiles are available for copying the bits of y laterally along their row, Westward. That is why we have used bidirectional copy tiles here. The bits are being passed along in two directions. In summary, the addition occurs along the main diagonal and flanking copy operations pass the data to the centre.

6.3 6-TAM Subtraction Tiles

Next, we define a system of 6-TAM tiles for computing the subtraction of two numbers x and y. The majority of our strategy will be the same as the addition

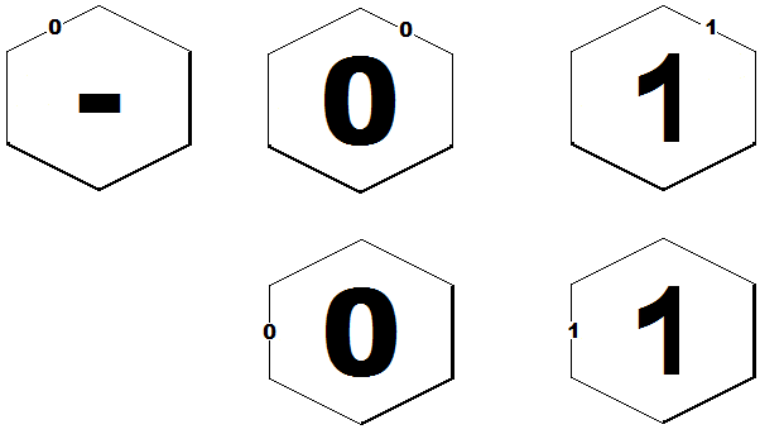

Figure 6.11: Hexagonal Subtraction System — Starting Tiles.

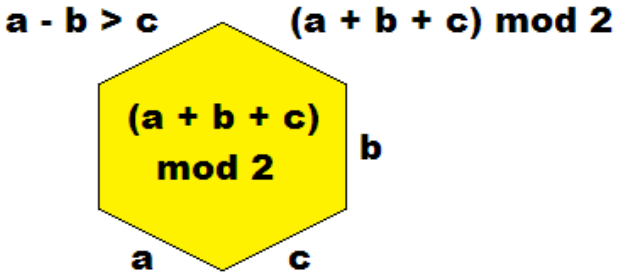

Figure 6.12: Hexagonal Subtraction System — Subtraction Tiles.

system example. Subtraction can be computed using the same bitwise, iterative technique of applying an operation on two bits from the input integers x and y and a carry bit from the previous steps output. The final solution is the number resulting from concatenating the outputs, in LSB to MSB direction.

Figure 6.11 defines the subtraction system starting tiles. The tile labelled '-' is new and has a 0 in the NW edge. Figure 6.12 defines the tiles that compute the

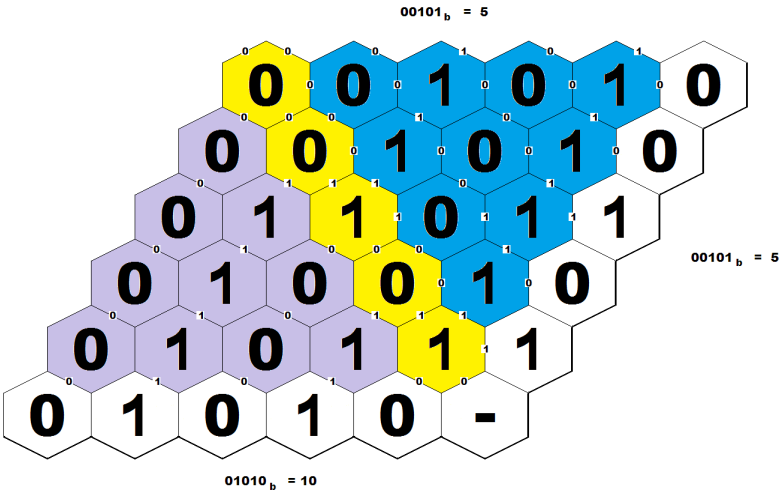

Figure 6.13: Hexagonal Subtraction Example.

subtraction. Formally the tile is defined as $\boldsymbol{t}_{-} = \langle \text{BLANK}, a - b > c, (a + b + c) \mod 2, b, c, a \rangle$ and $v(\boldsymbol{t}_{-}) = (a + b + c) \mod 2$, and $\forall a, b, c \in \{0, 1\}$.

Here, we use a generalized version of the greater than function from formula 6.2. Essentially, we take $a - b > c$ to return 1 when $a - b > c$, and return 0 otherwise. Figure 6.13 illustrates a completed final configuration computing the difference of the values $y = 00101, x = 01010$, and $y - x = 00101$. As with the addition system, we use the uni- and bidirectional copy tiles to pass information from the starting configuration tiles to the centre subtraction tiles residing along the main diagonal from SE to NW. The bidirectional tiles have the additional feature of passing data up rows in the NE direction, thereby providing an output along the top row.

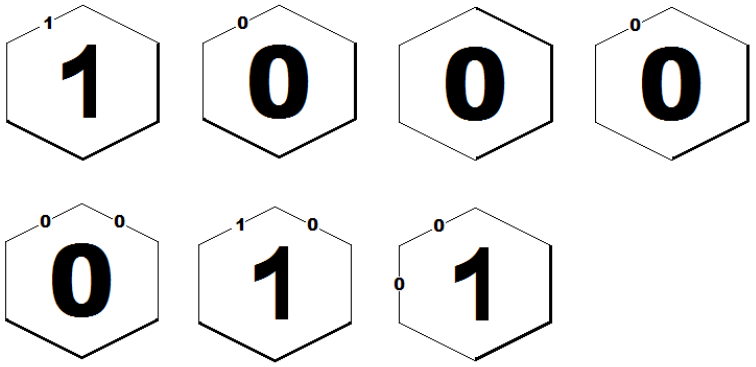

Figure 6.14: Hexagonal Multiplication System — Starting Tiles.

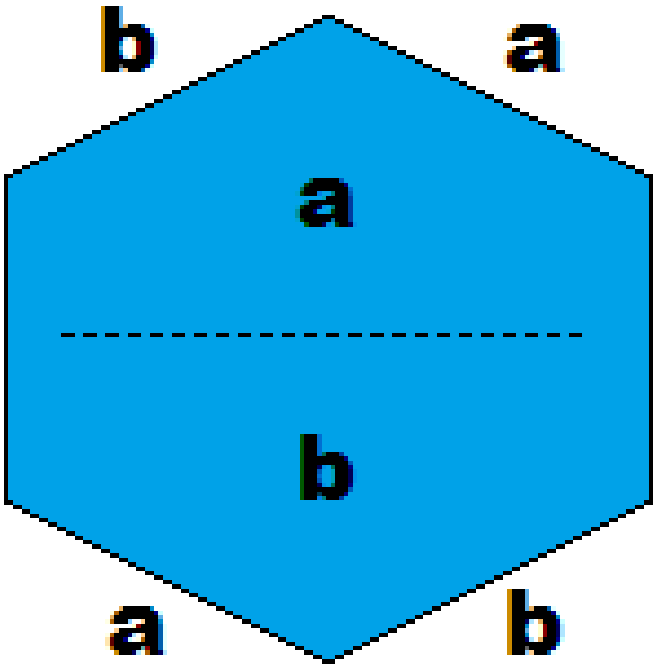

Figure 6.15: Hexagonal Multiplication System — Bitshift Tiles.

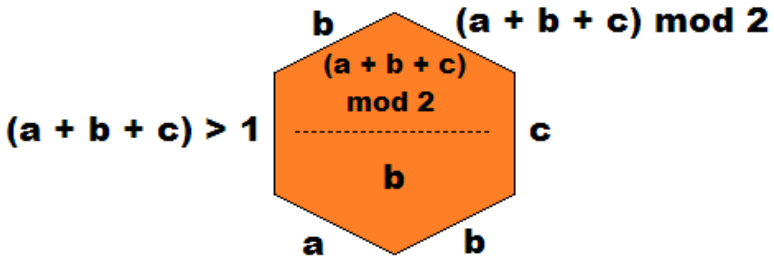

Figure 6.16: Hexagonal Multiplication System — Multiplication Tiles.

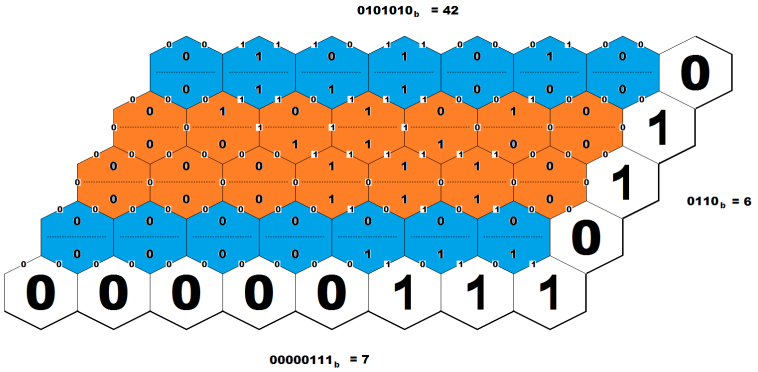

Figure 6.17: Hexagonal Multiplication Example.

6.4 6-TAM Multiplication Tiles

The next tile set we will define is for computing the multiplication of two numbers. This system is the most unique among the examples in this thesis. We provide the details of this system for two reasons. One reason is to demonstrate different techniques that are available in the literature. The second reason is to motivate the use of hexagons over squares for some solutions, because this example stands out as a major improvement over its square analogue. The multiplication system makes

use of two entirely new tile sets that have not been used previously in this chapter. Figure 6.15 shows the first such group of tiles we will define. We name the tiles from Figure 6.15 "bitshift" tiles and as the name implies, they perform a bitshift operation. We will describe the bitshift tiles in two steps. First we will describe how the tile works then we will show how bitshift is needed to compute the multiplication of x and y.

The tile depicted in Figure 6.15 is referred to as the bitshift. The formal definition of the bitshift tile is $\boldsymbol{t}_b = \langle \text{BLANK}, b, a, \text{BLANK}, b, a \rangle$ and $v(\boldsymbol{t}_b) = (a, b)$, for $\forall a, b \in \{0, 1\}$. The bitshift tile performs two functions at the same time: a copy operation from SW to NE and a copy operation from SE to NW. The latter copy, from SE to NW, is the bitshift operation in action. Since columns in this system are aligned SW to NE, the bitshift is copying the bit from row i to $i + 1$, but also from column j to $j + 1$.

Now consider the purpose of the bitshift tiles. In an algorithm for calculating multiplication of two binary numbers, $a = x \times y$, we use a combination of bitshift and addition. Recall the algorithm works by successive summation of the multiplication of x times successive digits of y i.e.,

$$a = \sum_{i=0}^{n-1} 2^i y_i x, \tag{6.3}$$

where n is the number of digits in y and the representation of digits is base 2. In base 10 this corresponds to exactly the long multiplication taught in elementary school. Refer to Figure C.1 for a comparison.

Figure 6.16 defines the second subset of tiles required to perform the binary mul-

tiplication algorithm. Similar to the bitshift tile, the multiplication tile performs two different operations at once. The first operation, same as before, is the SE to NW copy operation which is equivalent to a bitshift. The second operation is to sum the bits, $x_i + y_i +$ carry and output the sum (NE) and the carry (W). The formula for the multiplication tiles is $\boldsymbol{t}_\times = \langle (a+b+c) > 1, b, (a+b+c) \mod 2, c, b, a \rangle$, and $v(\boldsymbol{t}_\times) = ((a+b+c) \mod 2, b), \forall a, b, c \in \{0, 1\}$. Figure 6.17 shows how the bitshift and multiplication tiles work together and highlights their main difference. The bitshift tile works as special case of the multiplication tile. To understand Figure 6.17, recall the bitshift operation of both the bitshift and the multiplication tiles is recorded as the portion of label below the horizontal line on each tile. Therefore, in the completed final configuration, you can read subsequent bitshifts of the original value, $00\ldots0111_b = 7$, or the bottom half of the labels of rows, where each successive row above the first is another shift to the left. Next, consider the colour of the tiles. Each row will be only one colour and the colour is completely decided by the value of the starting (rightmost) tile in the same row: Blue is 0 and orange is 1. When implementing the long multiplication algorithm of Figure C.1, a 0 in bit i of the second integer corresponds to a row of all zeroes in the summation portion of the algorithm. Hence the row will not affect the total of the sum. This statement is reflected in the multiplication system by the blue tiles being defined such that they are a "no sum" version of the more general orange tiles. Meanwhile, the orange tiles do perform a row-wise summation which output the result to the row above. Therefore the final multiplication value can be read along the top row of the system.

Notice in Figure 6.17 the zero-padding of the bottom input is quite long. This of

course is because the multiplication of two's complement integers $x \times y$ can cause an overflow error if the number of bits allowed in the output is not larger than or equal to $\log_2(y) + \log_2(x)$. In other words, if the inputs can be encoded by a and b number of bits, then the height must be b and the width must be $a + b$.

6.5 6-TAM Addition with more than two operands

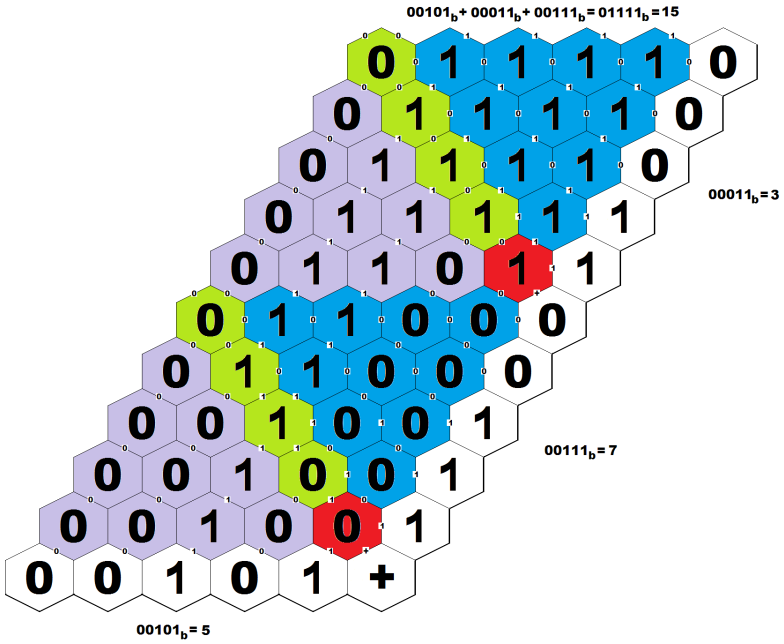

Figure 6.18: Hexagonal Addition Three Operand Example.

Now we will continue the discussion for arithmetic tile systems by introducing a couple of useful variations that are possible. We will use these variations in the next chapter as they are fundamental techniques for solving more complex computations, and in particular NP-Complete decisions problems. Figure 6.18 shows the strategy

commonly used for the computation of more than two values. In this example, we use the tiles from T_+ to construct a starting configuration encoding three inputs. As associativity applies, we can place the inputs arbitrarily without loss of generality. The resulting final configuration uses twice as much space as the previous example from Figure 6.10. This is because the sum is clearly being applied twice. Furthermore, recall from the previous addition examples that the bits are padded with zeros to avoid overflow errors. As such, each subregion will be the same size as the one above or below it. Or in other words, all subregions that are attached will have equal width and height, both respect to themselves and each other.

Note the colour-coding of the tiles provides a clear visual cue that two additions are being performed. The red tile, adjoined by a diagonal length of green tiles is the most prominent feature demarcating the addition region.

In the general case, we define the area performing one operation as a super tile. We can then reason about more advanced systems at a higher level using the concept of super tiles to aid in design and visualizing the data flow. For example, Figure 6.18 has two super tiles, each performing an addition operation. The output of the first super tile feeds as input into the second. In the general case, an assembly's design allows the combination of operations in a more complex system. We will demonstrate examples of that now as well as introducing one more concept, non-deterministic tiles.

6.6 6-TAM Addition With Non-Deterministic Tiles

The next set of tiles we will define moves the discussion from how to compute a single operation to how to solve a problem requiring the computation of multiple

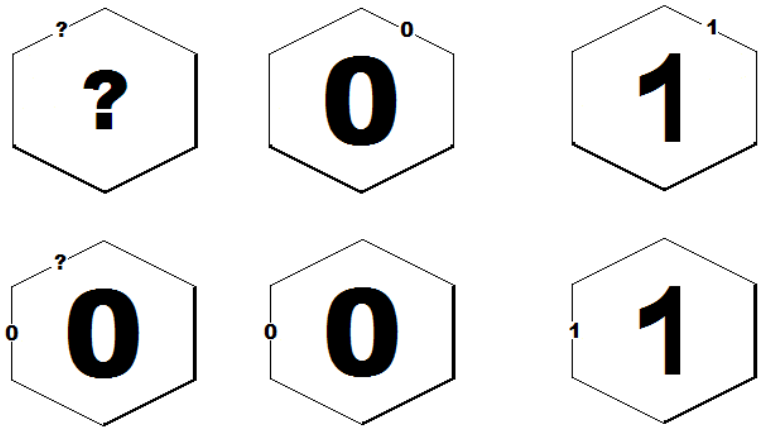

Figure 6.19: Hexagonal Non-Deterministic Addition System — Starting Tiles.

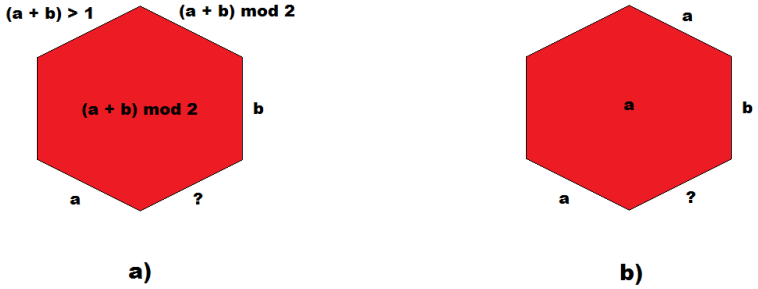

Figure 6.20: Hexagonal Non-Deterministic Addition System — Non-Deterministic Add/Copy Tiles.

operations. When tile assembly model algorithms are required to make choices, this is implemented by use of the non-deterministic order and selection of 6-TAM tiles attaching at each assembly step. Therefore, a non-deterministic algorithm can be modelled in 6-TAM by defining tiles such that at certain steps of the assembly, more than one tile may be eligible to attach to a configuration at the same position. We will now illustrate this concept with the following example which will also be useful

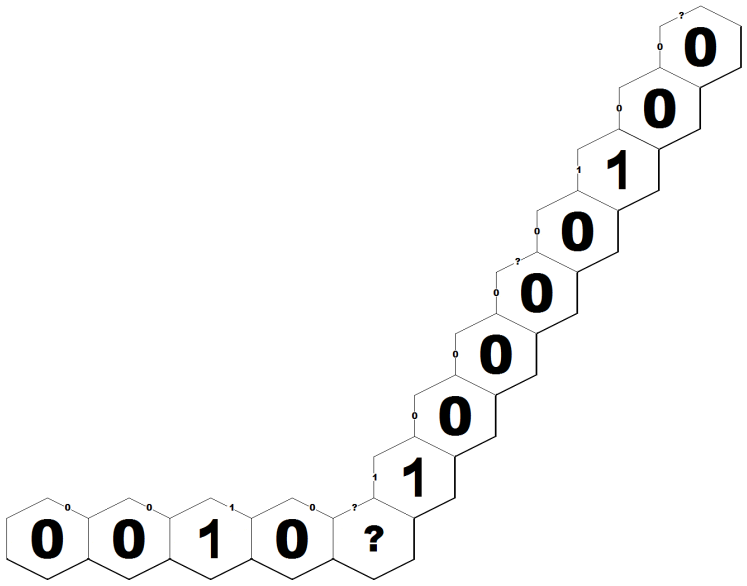

Figure 6.21: Hexagonal Non-Deterministic Addition System — Sample Start Configuration.

for understanding the 0-1 Knapsack solution in Chapter 7. These non-deterministic tile sets will model the algorithm of choosing a subset to sum. As discussed, each element in the subset will be chosen non-deterministically.

Figure 6.19 defines the starting tiles for the non-deterministic addition system. The tiles are similar to the previous addition system, however the right-hand side can now assemble additional rows, with more than one input on that side. Figure 6.21 shows an example of a start configuration with two inputs on the right-hand side. Notice the tiles extending along the diagonal to the North-East direction. After the '?' tile, there are four more tiles for the first input, then four more for the second input. The fourth tile is labeled 0 but also has a '?' on its North-West edge, so that it can be used as both a 0 and '?' tile. One more thing to note is that we don't define

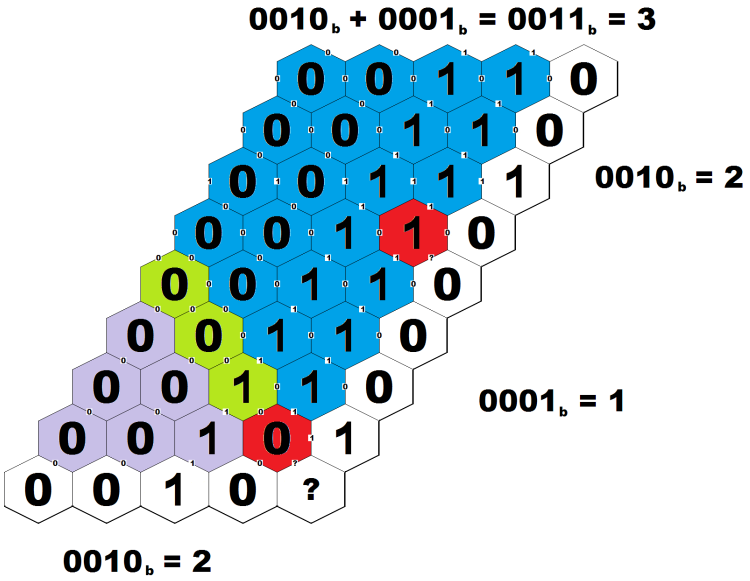

Figure 6.22: Hexagonal Non-Deterministic Addition System — Sample Non-Deterministic Final Configuration #1.

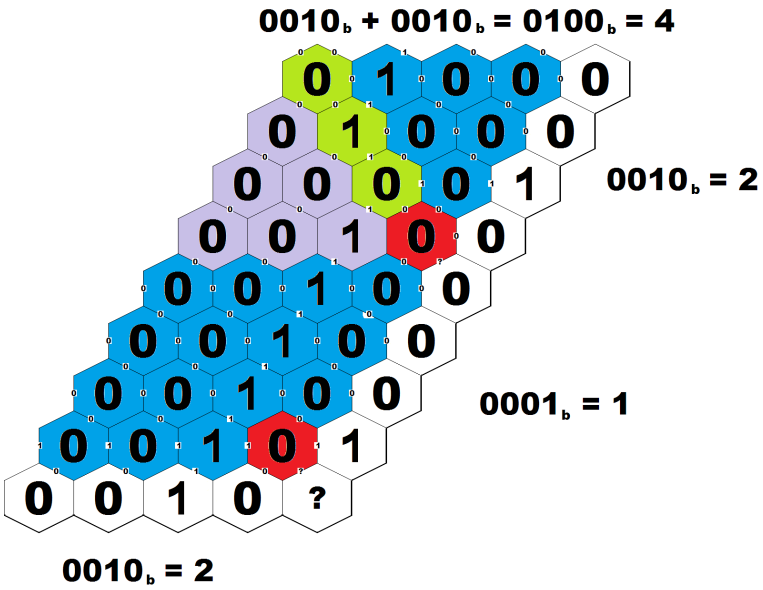

Figure 6.23: Hexagonal Non-Deterministic Addition System — Sample Non-Deterministic Final Configuration #2.

any tiles labelled 1 and with '?' on it's North-West edge. This is simply because we do not allow any negative inputs since the representation used here is two's complement.

Figure 6.20 defines two new sets of tiles. Tile (a) is the addition tile from before, modified to have a '?' symbol instead of a '+' symbol. Tile (b) is the unidirectional copy tile from before with again the same modification: The SE edge has been replaced with a '?' symbol. Note the definition of these tiles is $\boldsymbol{t}_{?+} = \langle \text{BLANK}, (a+b) > 1, (a+b) \mod 2, b, ?, a \rangle$, and $\boldsymbol{t}_{?c} = \langle \text{BLANK}, \text{BLANK}, a, b, ?, a \rangle, \forall a, b, \in \{0, 1\}$. The E, SE and SW edges are ambiguous between the two sets of tiles, (a) and (b). For any values $a, b \in \{0, 1\}$ there exists two tiles that may fit at the position with neighbouring binding domains a, ?, and b, respectively. Recall that final configurations for an assembly system are not required to be unique. Therefore for each such ambiguous position, the final configuration produced will be non-deterministically chosen between two outcomes.

For example the starting configuration with two ? tiles, such as the one depicted in Figure 6.21, will non-deterministically produce one of $2^2 = 4$ final configurations. Figures 6.22 and 6.23 demonstrate two possible outcomes of this scenario. When the assembly starts with the starting configuration in Figure 6.21, either addition or copy will be chosen for each operation. Therefore the values of the tiles on the top row can be any combination of summing, or not, the values encoded on the right-hand side of the starting configuration. To clarify, Figure 6.22 is the result when the first input is non-determinisitcally chosen as addition and the second input is non-deterministically chosen as a copy operation. For short, we say the first input was "chosen" to be in the sum. Similarly, it can be said the second input was not chosen. The second example,

in Figure 6.23 is opposite. The second variable was chosen, but the first input was not. Therefore, note the total sum, as read by the bits in the top row of either final configuration, is different.

In the next chapter we will introduce additional techniques to apply verification to the non-deterministic steps chosen, therefore combined with the operations we've already defined, we will give a technique for solving NP-Complete decision problems.

Chapter 7

6-TAM Tile Definitions for the 0-1 Knapsack Problem

7.1 Tile Definitions

In this chapter, we will first introduce the tile sets to solve the 0-1 Knapsack problem, then we will show the solution assembled from these tiles. Figure 7.14 illustrates an overview of what will be accomplished in this chapter. Figure 7.14 depicts a final configuration that solves an instance of 0-1 Knapsack problem using the tiles defined in this chapter.

Recall the definition of 0-1 Knapsack from Chapter 4. The problem defines a set of objects each with a capacity and a value. The goal is to find a subset of objects whose capacities sum to less than a constant, W, and whose values sum to greater than a constant, V. In order to define the tile set to solve the 0-1 Knapsack problem, we break up the problem into smaller logical pieces, which we will be calling 0-1 Knapsack

tile subsets. Similarly, when the tiles of one subset are defined in a contiguous region, we will refer to that region as a subregion. Then, using the results we've defined in Chapter 6, we will then show how to combine each of the subsets by assembling the various subregions to calculate the solution to 0-1 Knapsack problems.

First, we will give an overview of the different tile subsets we will use and then define them, describe how they work, and most importantly, how they combine to solve the 0-1 Knapsack problem. Before that however, recall the 0-1 Knapsack problem requires the sum of two quantities; a capacity and a value. Similar as Wang et al.'s [16] solution of 0-1 Knapsack using square tiles in Chapter 4, we will use a U-shape starting configuration. Therefore, there will be a symmetry to the tile definitions, as we'll describe tiles that join on the left hand side as well as the right hand side. A majority of the time, the tiles that attach to the left will be the mirror image of the tiles on the right. Recall the definition of the mirror image from Chapter 2 as it will be used frequently in these definitions.

The first tile subsets we will define are the Copy tiles: the Unidirectional and Bidirectional Copy tiles. Both the Uni- and Bidirectional tiles were defined previously in Chapter 6.2, in the section on Addition tiles. The Bidirectional copy tiles are also used to represent a non-deterministic choice of the current element not being chosen in the subset sum. Alternately, the second tile set we will define is the aforementioned Addition tiles, which are used to represent when an element is chosen to be in the subset sum. Next, we define the Non-deterministic tiles. These are the tiles that catalyze the choice between "in the subset" and "not in the subset", therefore they share a strong relation with the aforementioned Addition and Bidirectional copy tiles

from Figure 6.20.

In addition to the subsets required for deciding the inclusion of elements in the knapsack, we define several subsets for the purpose of synchronizing the non-deterministic choices, validating algorithmic correctness, and verifying the result of the sums in the final configuration. Collectively, we refer to the subsets in this category as "Verification" tiles.

The first set of verification subset we define are the Lateral Synchronization verification tiles. The purpose of these tiles is to fill the space in between the left and right hand side subset sums, but only if the non-deterministic choice for each element on the right matches the choice made for the element on the left. The second verification subset is the Greater Than or Equal tiles. These tiles are designed to attach to the configuration after a subset sum on the right hand side has been decided by there being a choice for all elements. Then these tiles will verify the value criteria of the 0-1 Knapsack was satisfied by testing if the subset sum on that side is greater than or equal to zero. After those, we define the Less Than Zero subset. These tiles have the similar purpose as the Greater Than or Equal to Zero tiles, except these are used to ensure the capacity restriction is met. The next tile subset is called the Top Row tiles. These tiles only have the purpose to fill the positions on the top row between the left hand and right hand sides. Finally, the Checkmark subset is defined. The purpose of the Checkmark tile is that one will attach to the final configuration if and only if all other verification tiles have successfully verified all other restrictions and criteria to be true. Therefore, the existence of a Checkmark tile in a final configuration implies the solution for the 0-1 Knapsack problem instance encoded by that configuration's

starting configuration.

7.1.1 Starting Tiles

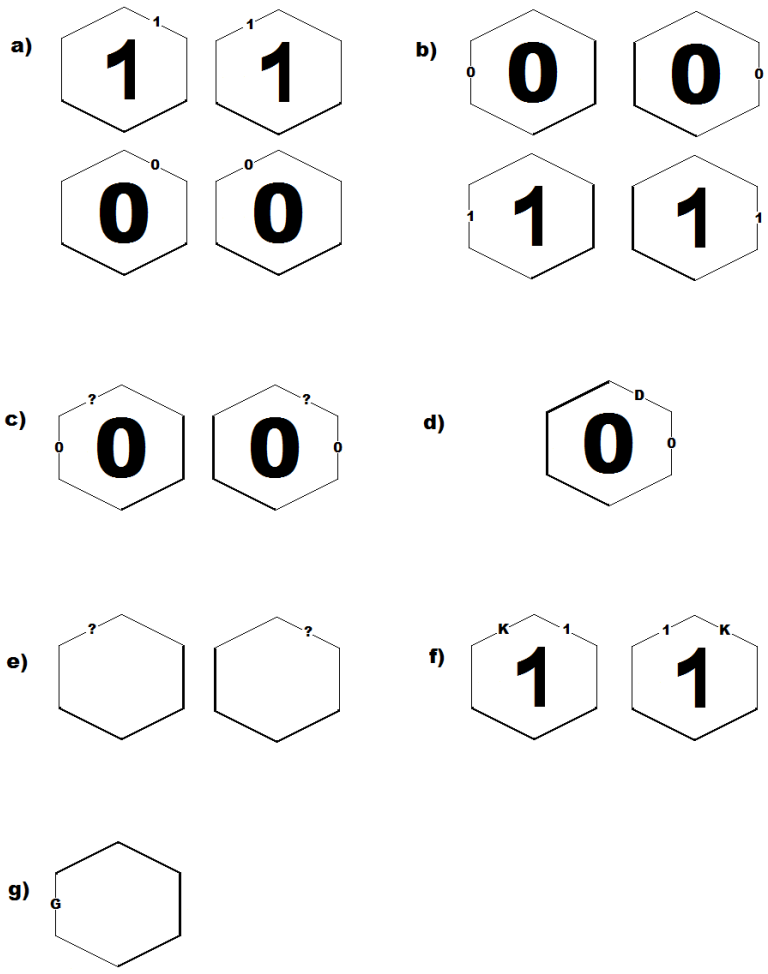

Figure 7.1: Starting Tiles

The starting tiles are used to create the initial seed configuration. They are not included among those tiles that can attach to the current configuration during the assembly process. They are also not subject to the constraints given by definitions of

Chapter 2 as they are not left to self-assemble naturally but through intervention by the experimenter.

The formal definitions for the starting tiles used by the 6-TAM solution of the 0-1 Knapsack Problem are defined as $\boldsymbol{s}_i$ in Appendix B.1 and Figure 7.1 provides an illustrative definition.

7.1.2 Undirectional Copy Tiles

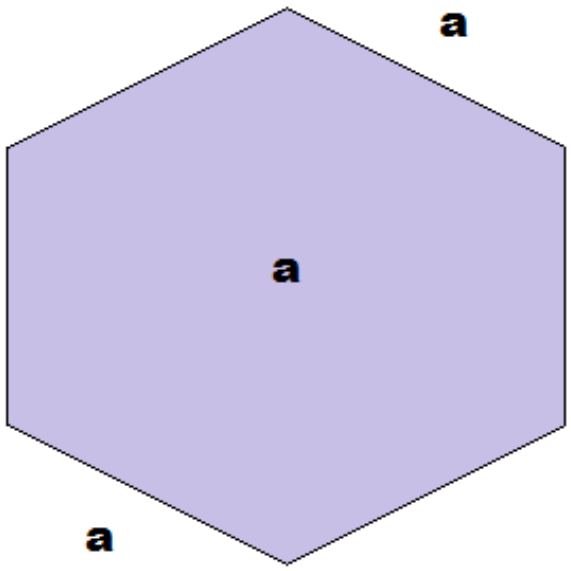

Figure 7.2: Unidirectional Copy Operation

Next, we'll define the non-starting configuration tiles. First in this category is

the Unidirectional Copy Tiles, T_1. These tiles are precisely the same as those that where described in Chapter 6, with the addition of the mirror images of those tiles. Figure 7.2 provides an illustrative definition of these tiles, $\boldsymbol{t}_1(a) \in T_1$. Refer to Appendix B.2 for the formal definitions. The Unidirectional copy tiles are used to pass along data in one direction. In the 6-TAM solution for the 0-1 Knapsack problem, we require such tiles for copying the data from a bottom-to-top direction. Bottom-to-top corresponds to South-West-to-North-East for tiles in the right half of the final configuration and South-East-to-North-West for tiles in the left half. The Unidirectional copy tiles have the property that their neighbours along a row (West-to-East) do not pass along or receive any additional information.

The single pair of tiles $\boldsymbol{t}_1(a)$, given by enumerating a over $\{0, 1\}$ and their mirror images $\overline{\boldsymbol{t}_1}(a)$ are defined in this subset T_1 and are elements of T.

7.1.3 Bidirectional Copy Tiles

Figure 7.2 and 7.3 were previously shown in Chapter 6 and are defined in Appendix B.2. They remain unchanged and follow the same reasoning for existing in this system. They are ideal for transferring data in linear paths from one side of a subregion to another. In particular, they have specific use in the addition subsystem for copying values in within a row as well as between rows. See Figure 6.10 to recall the addition example.

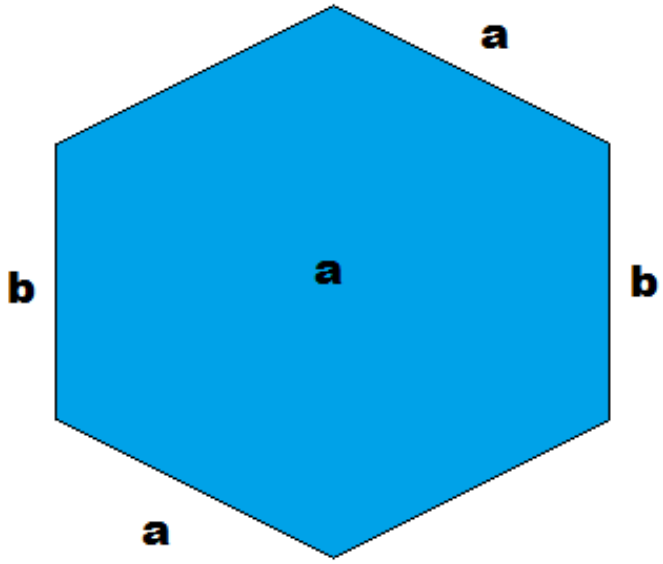

Figure 7.3: Bidirectional Copy Operation

7.1.4 Bidirectional Copy Tiles with Verification Inputs

The tiles $\boldsymbol{t}_3 \in T_3$, given by Figure 7.4 and formally defined in Appendix B.3, introduce a new pair of symbols, '+' and 'C'. The figure shows a new parameter, and the tile is defined for each value of $\alpha \in \{+, C\}$. The tiles with the '+' and 'C' symbols are designed to fit on the bottom row of copy and addition subregions. They act to synchronize the left and right sides of the U-shape starting configuration. Naturally, the '+' symbol will attach when the non-deterministic tile in the same row is labelled '+' and the same is true with respects to the 'C' symbol. In the context of the whole configuration, it's possible that the item in the knapsack was chosen on the left hand

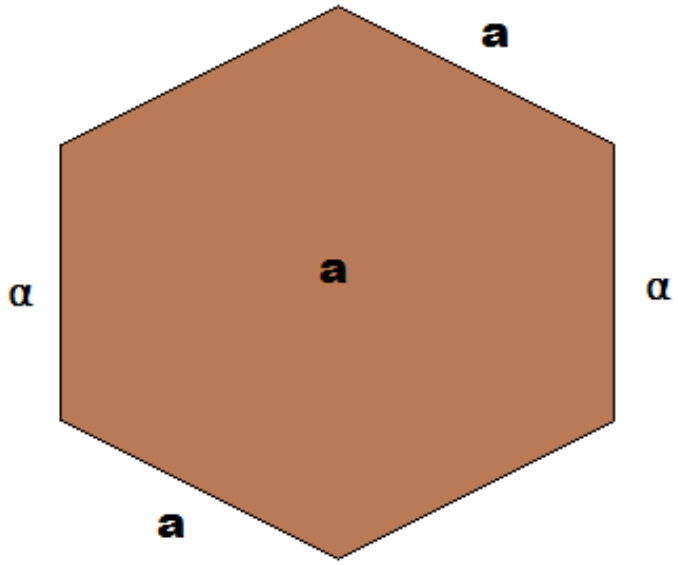

Figure 7.4: Bidirectional Copy Operation with Verification Inputs

side, but the same item was not chosen on the right hand side. In other words, an addition operation is being performed on the left hand side and a copy operation is being performed on the right hand side in the same rows. This is why the $\boldsymbol{t}_3$ tiles are important. In this scenario, the verification tiles will not fill every position in that row, since the '+' and 'C' symbols will not be adjacent edges, therefore the final configuration will have holes and the verification has failed. See Section 7.2 below for more details on interpreting the scenarios when verification of subregions fail.

7.1.5 Addition Tiles

The Addition tiles, $\boldsymbol{t}_4 \in T_4$, are exactly the same as defined in Chapter 6, are shown here in Figure 7.5, and formally defined in Appendix B.4. We use the same definition and its mirror image. Formula B.5, in Appendix B.4, defines the complete

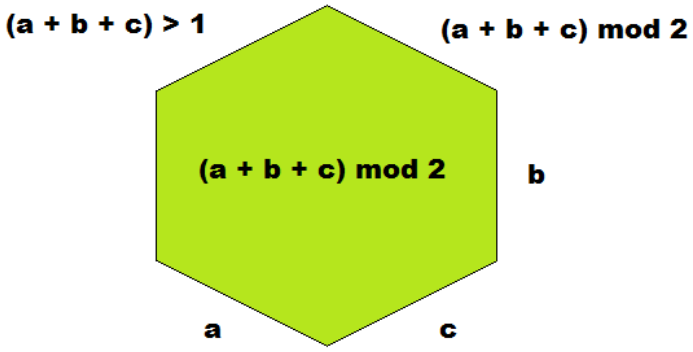

Figure 7.5: Addition Tiles

subset. The addition tiles have exactly the same semantic meaning and same usage as before. They compute the sum and carry of the bits x_i and y_i and the carry from the previous sum. The pattern follows simple long-form arithmetic used to sum two integers represented in binary notation.

7.1.6 Non-deterministic Tiles

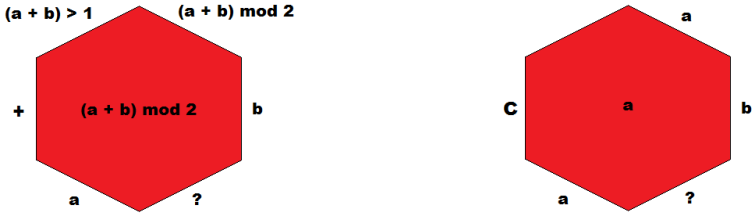

Figure 7.6: Non-Deterministic Tiles

The non-deterministic tiles, $\boldsymbol{t}_{5_i} \in T_5$, illustrated in Figure 7.6 and formally defined in Appendix B.5, serve identical purpose to the tiles of the same name defined in Chapter 6, but with some minor modifications to allow them to fit within the 0-1 Knapsack solution (which are in turn modifications of the addition and bidirectional copy tiles). The addition tiles have added the '+' symbol to their West edges. Alternately, the copy tiles have added the 'C' symbol. Both sets are also defined to include their mirror image. The '+' and 'C' symbols are required in the first row of each '+' or 'C' super tile subregion. These tiles are meant to be the first tiles to assemble in the first column of the first row to assemble, however there is a scenario examined below, in Section 7.2.2, where these tiles could end up not being first to assemble in a row.

Notwithstanding this edge case scenario, it is still correct to logically reason about the tiles as if the first tile to assemble will be the non-deterministic tile. Therefore, they decide the operation performed in that subregion. Every other tile assembled in the subregion will comply by also being a member of the appropriate subset performing the operation if and only if the final configuration is positive. Moreover, the left and right halves of the U-shape must comply with the non-deterministic tile on the left being of the same operation as the non-deterministic tile on the right of the same row if and only if the final configuration is a positive output. This synchronization is logically ensured by consequence of the true positive if and only if the system is defined properly within the subregion by tiles in Figure 7.4 and within the space between the left and right columns of super tiles by the tiles defined in Figure 7.7.

7.1.7 Lateral Synchronization Verification Tiles

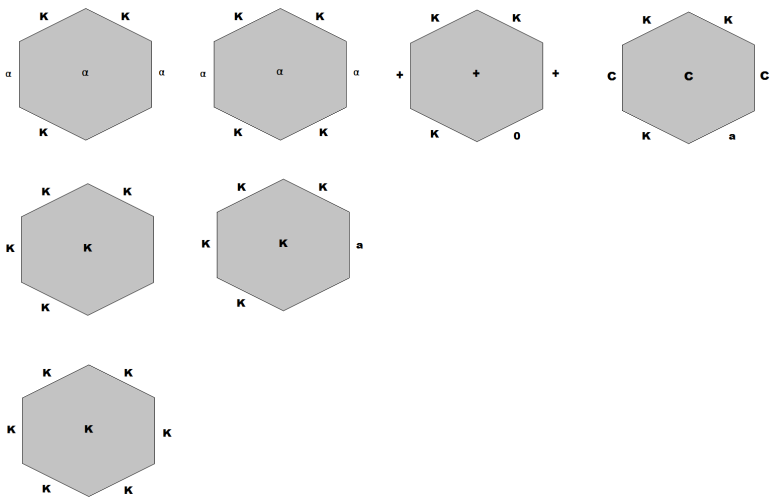

Figure 7.7: Lateral Synchronization Verification Tiles

The next tiles defined are $\boldsymbol{t}_{6_i} \in T_6$, illustrated in Figure 7.7 and formally defined in Appendix B.6. These tiles are defined to copy the 'C' and '+' symbols laterally. Note the parameter α in Figure 7.7 represents a tile defined for $\alpha \in \{C, +\}$. The symbol 'K' is introduced and represents no specific meaning or value. Simply, the 'K' symbols are strategically defined on edges so as to make the tiles fit in their region (middle verification area) and to make them uniquely defined on the East, West, South-East, and South-West edges, compared to tiles in other subsets. Take for instance tile $\boldsymbol{t}_{6_3}(0) = \langle C, K, K, C, 0, K \rangle$ and note the East, West, South-East and South-West edges are in order: C, C, 0, and K. Now recall the Bidirectional copy tile $\boldsymbol{t_3}(0, C) = \langle C, \text{BLANK}, 0, C, \text{BLANK}, 0 \rangle$ and note its East, West, South-East and South-West edges in order: C, C, 0, and BLANK. Therefore, these two sample tiles differ on those edges by only a K that exists on the tile $\boldsymbol{t_{6_3}}$, meaning if the K symbol

did not exist, there would be ambiguity and these tiles from separate subsets would be chosen to attach to this position non-deterministically. This outcome needs to be avoided, hence, the symbol K was introduced to make those edges unique. Finally, the other values for the lateral synchronization tiles are 0 and 1. These values are placed on edges that are exposed from the left and right regions to the middle verification region.

Mirror images exist in T_6 for $\boldsymbol{t}_{6_1}$, $\boldsymbol{t}_{6_2}$, ..., $\boldsymbol{t}_{6_6}$. Tile $\boldsymbol{t}_{6_7}$ is its own mirror image $\boldsymbol{t}_{6_7} = \overline{\boldsymbol{t}_{6_7}}$, therefore it is included only once because by the definition, tile definitions are unique.

7.1.8 Greater Than Or Equal To Zero Verification Tiles

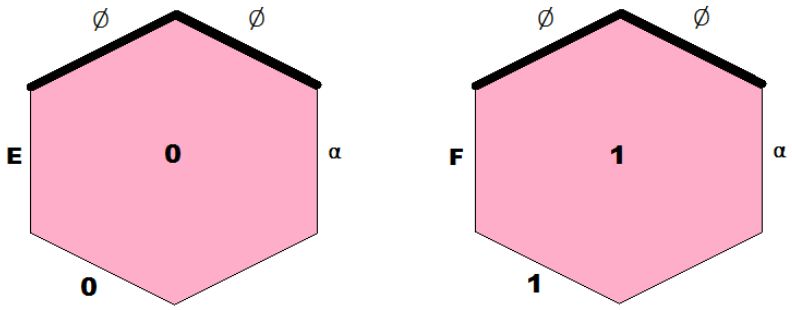

Figure 7.8: Greater Than Or Equal To Zero (Right Hand Side) Verification Tiles

Next, we will explain the "Top row verification" tiles. We want to describe the logic of these tiles differently than before. Because these tiles are the top row, they are guaranteed to be the last to assemble and we can use that assumption as we reason about them. Also they assemble on a single row, in a single direction, one

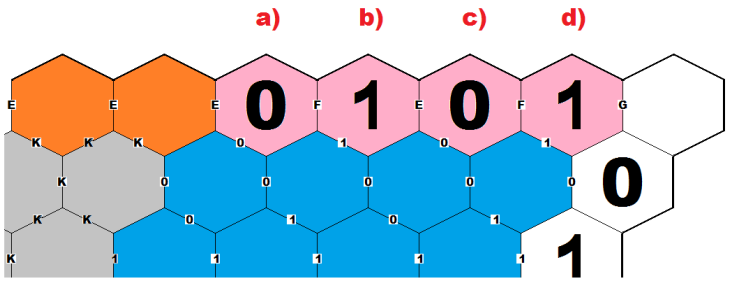

Figure 7.9: Greater Than Or Equal To Zero — Example

tile at a time. Since they have a deterministic ordering and assemble discretely, they are logically equivalent to finite state machines where the final state corresponds to a "yes" or "no" result of the verification process. The generalized pattern of verification subsystems is to decide yes or no if a set of inputs on the top edge, when taken in linear order, are a valid output to the subregion. The tiles are defined such that a confirmed "no" output will leave the final configuration without a checkmark tile. Hence, an invalid output of a subregion yields a negative instance decided for the particular assembly instance. The tiles $\boldsymbol{t}_7 \in T_7$, shown in Figure 7.8 and formally defined in Appendix B.7, are called "Greater Than or Equal to Zero" verification tiles. Observe the paramater α in this figure defines the tiles once for each of the symbols $\alpha \in \{E, F, G\}$. Figure 7.9 provides an example of these verification tiles in one possible final configuration and illustrates the purpose of these tiles. Note that unlike other tiles defined in this chapter, there exist no mirror images these tiles in T_7. As the name describes, these tiles verify that the output, along the top row of the right region is greater or equal to zero. The goal of this region is to maximize the value of items chosen in the knapsack. The values were added as positive numbers,

however the starting value encoded in the start configuration was negative. Therefore the output of the summation is positive if and only if the total of chosen values is greater than the absolute value of the starting value.

The low level design of the tiles in Figure 7.8 is simply to create a subsystem that will assemble if and only if the MSB is 0. These tiles introduce the symbols 'G', 'E', and 'F'. These particular symbols were chosen arbitrarily only because they were not yet used by other tiles. 'G' is the symbol that is used by the starting configuration and therefore it is shown in Figure 7.9 as tile (d) and it is always the first tile to join the assembly in the position immediately one space West of the starting configuration tile. The tiles are designed to verify that the binary value given by the North-East edges of the top row on the right side of the assembly is positive with the MSB on the left and the LSB on the right. Therefore, a binary value encoded as a two's complement integer is only positive if the MSB is 0. Since the other bits don't affect the output, it's the last tile that is most significant, in the sense that if the last tile has 'E' on its West binding domain, it's joined to a 0 and hence 0 was the MSB. This position is labelled with tile (a) in Figure 7.9. If the last tile was an 'F', then the tile has 1 on its South-West edge and the MSB was 1, indicating the value of the right summation was negative. The tile in position (b) illustrates this point. Note that the bit below the top row is a 1. However, since it is not the significant bit, it is not the bit that decides the sign. By design of the system, and in part by this subset, when the value of the subset sum is negative, then the checkmark tile will not assemble. This is ensured on the right hand side by the tiles in Figure 7.8 because no other tile defined will assemble West of the tile with an 'F', if it was in the last position.

7.1.9 Top Row Verification—Transmission Tiles

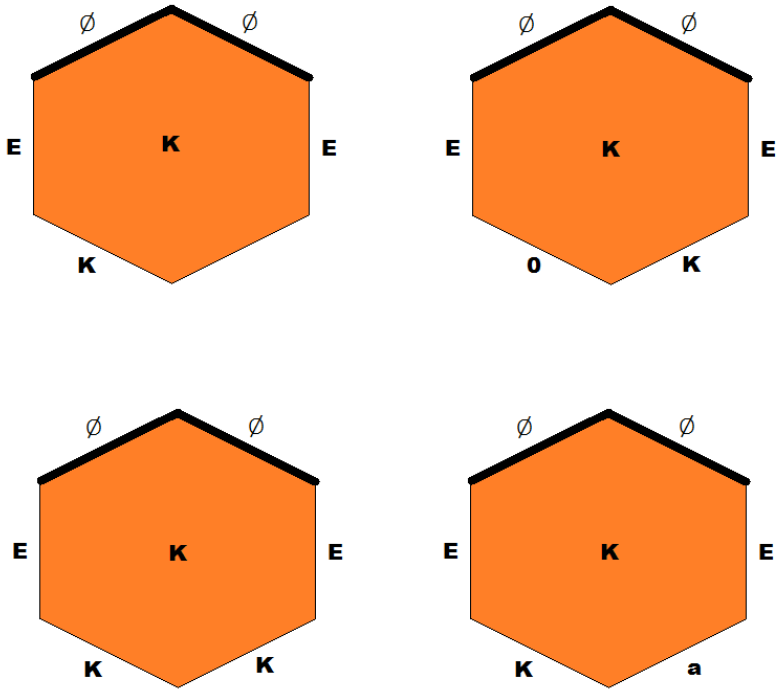

Figure 7.10: Top Row Verification—Transmission (Centre) Tiles

The top row verification transmission tiles, $\boldsymbol{t}_{8_i} \in T_8$, illustrated in Figure 7.10 and defind in Appendix B.8, are very simple in their purpose. They will fill the top row above the centre verification region if and only if the previous verification tiles ended with an 'E' on the West edge. Essentially, the tiles serve to fill the region between the left and right sides so they are defined based on what fits in that region. Specifically, the West and East edges on all tiles are 'E' because the tile from Figures 7.8 and 7.10 expect 'E' when they are verified as correct. One of the tiles has an important role, $\boldsymbol{t}_{8_3}$, because it has a 0 on the South-west edge. This tile will therefore only assemble

if the MSB of the left side summation (right-most bit) is 0. Moreover, there is no tile $\boldsymbol{t}_{8_i}$ that will assemble adjacent to the MSB if it is 1. The North-East edge of the tile assembled in the MSB position of the top of the left side region could be one of three possibilities:

a. BLANK (copy Tiles)

b. 0 (addition tile, no carry)

c. 1 (addition tile, with carry)

The tiles in Figure 7.10 are defined such that there is no tile to assemble in scenario (c).

7.1.10 Less Than Zero Verification Tiles

The tiles $\boldsymbol{t}_{9_i} \in T_9$ in Figure 7.11 are called "Less than Zero (LHS) Verification" tiles. Their formal definition is provided in Appendix B.9 and Figure 7.12 depicts an example of how these tiles are used in a final configuration. As their name implies, they serve to verify the output of the summation of the left side region is a negative number. Recall that the LHS represents the weight of the items in the knapsack. We begin with weight capacity encoded as $-W$ in two's complement. Therefore, if the sum total $\Sigma w - W \leq 0$, then $\Sigma w \leq W$. Therefore, testing if the total is positive is equivalent to testing if the chosen objects in the summation exceed the weight capacity. If the weight capacity was exceeded, then the verification should fail, meaning the assembly will be at a final configuration before the checkmark tile can assemble.

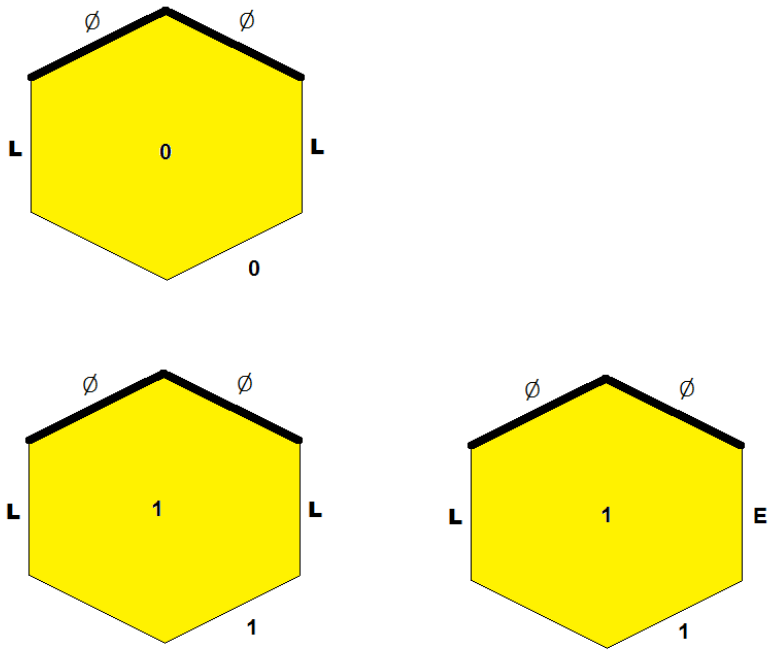

Figure 7.11: Less Than Zero (Left Hand Side) Verification Tiles

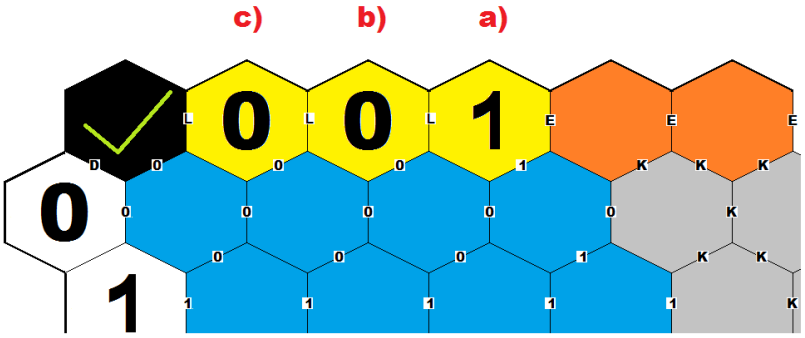

Figure 7.12: Less Than Zero — Example

The tiles accomplish this verification by assembling if and only if the LHS output is negative, i.e., The MSB is 1. Tile $\boldsymbol{t}_{9_3}$ is the important piece for this step, as it is the only tile that can be the neighbour of the Figure 7.11 tiles. This is because

it has 'E' on its East edge and it is therefore the only tile that is North-west of the MSB. Figure 7.12 illustrates the point. Note the tile in the position labelled (a) has an E on its East edge and a 1 on its South-west edge. The 1 on the South-east edge of $\boldsymbol{t}_{9_3}$ ensures the MSB is 1. The other two tiles in positions (b) and (c) are simply defined to assemble along the remainder of the top row, regardless of the bits along the output, since only the MSB of a two's complement value is used to determine its sign. Therefore, the South-west edge may be a 1 or 0 as both variation of these verification tiles are defined.

Note that this subset of tiles contains a total of only three tiles. There is no enumeration necessary, nor is there any mirror images defined.

7.1.11 Checkmark Tiles

Finally, the checkmark tiles $\boldsymbol{t}_{10} \in T_{10}$ in Figure 7.13 will assemble to make the final configuration if and only if all the verification systems assembled properly. Refer to Appendix B.10 for the formal definition of these tiles. The presence of the checkmark tile in the final configuration establishes that the instance of 0-1 Knapsack is a "Yes"-instance. Namely, this indicates that there exists a choice of $\boldsymbol{x}$ such that the capacity is less than W and the value is greater than V. Note that if the assembly did not contain a checkmark, it does not mean no such assignment exists. The TAM has the advantage of creating arbitrarily large number of assignments therefore only the existence of one or more positive results out of all choices needs to be identified to achieve the result.

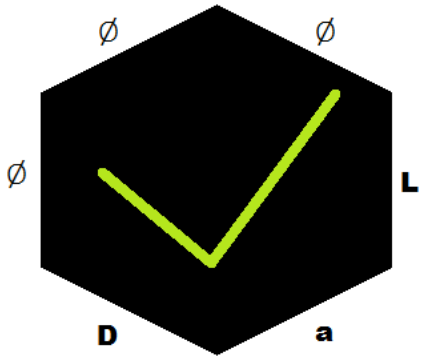

Figure 7.13: Checkmark Tiles

7.2 Possible Final Configurations

Three distinguishable outcomes are possible when the 6-TAM 0-1 Knapsack Tile produces a final configuration. In this section, we describe the distinct characteristics of the three results, how they are produced, and how the existence of said results contribute to the algorithm's final output when many instances of the assembly are produced non-deterministically and in massive quantities.

We begin by defining the categories which encompass the set of all final configurations. The following three categories are possible:

- True Positive
- True Negative
- False Negative

What follows in the sections below are particular examples and explanations for the 6-TAM 0-1 Knapsack solution. Where observations can be generalized to other problems, other degrees of TAM, or both, it will be noted explicitly.

7.2.1 True Positive

There is a single, distinguishable outcome in the True Positive categorization, namely when the final configuration yields a correct solution to the problem instance, and therefore contains the checkmark tile in its final configuration. Note the set of final configurations containing the checkmark tile for a given final configuration produced by a given starting configuration is not required to be a singleton. Since

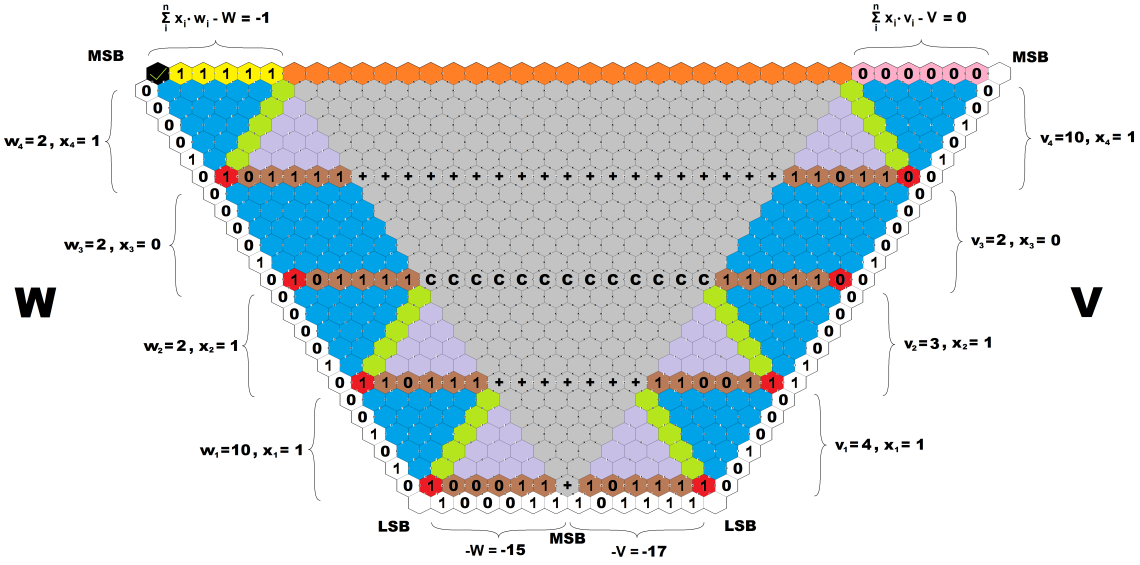

Figure 7.14: Hexagonal Tile Assembly Model—0-1 Knapsack Solution—True Positive

NP-Complete problem instances do not require unique solutions, tile assembly model assemblies that solve those problems also need not require unique solutions. For example, Figure 7.14 illustrates one such final configuration that was produced from the starting configuration that encoded the following 0-1 Knapsack instance.

$$\boldsymbol{v} = \langle 4, 3, 2, 10 \rangle, \tag{7.1}$$

$$\boldsymbol{w} = \langle 10, 2, 2, 2 \rangle, \tag{7.2}$$

$$W = 15, and \tag{7.3}$$

$$V = 17. \tag{7.4}$$

The figure illustrates a solution instance that solves the problem for these inputs, namely, $\boldsymbol{x} = \langle 1, 1, 0, 1 \rangle$. Therefore, the first, second, and fourth item was added to the knapsack. The sums of the first, second, and fourth elements in $\boldsymbol{v}$ sum to 17,

and the corresponding elements in $\boldsymbol{w}$ total 14. Hence, the values in the knapsack satisfy the goal of the 0-1 Knapsack problem. The chosen value is equal or greater than $V = 17$ and the weight of those objects does not exceed $W = 15$. Therefore, the checkmark tile assemblies in this final configuration and the entire self-assembly process will yield a positive result when the checkmark tile is detected in the set of final configurations.

7.2.2 True Negative

There are four possible variations on True Negative. We assign the following names to each, then we will discuss each in detail.

a. Fail by first verification,

b. Fail by second verification,

c. Fail by variables not matching, and

d. Fail by verification tiles being chosen non-deterministically.

Fail by first verification

The final configuration in Figure 7.15 shows an incorrect solution for this instance of 0-1 Knapsack. The items chosen for the knapsack is the first and fourth items, $\boldsymbol{v}_1 = 6, \boldsymbol{w}_1 = 5, \boldsymbol{v}_4 = 2$ and $\boldsymbol{w}_4 = 2$. Therefore, the weight constraint is satisfied since $5 + 2 - 10 = -3 < 0$, but the value constraint is not since $6 + 2 - 12 = -4 \not\geq 0$. Since one of the constraints was not satisfied, we expect not to see the checkmark tile in this configuration, which is true. The reason the checkmark cannot assemble to this

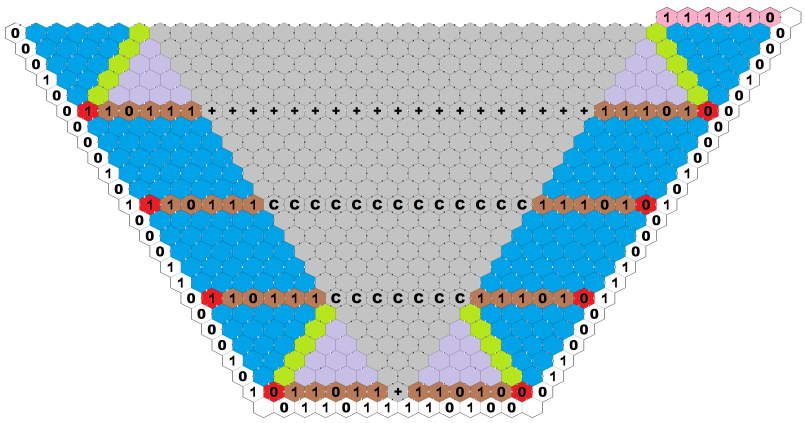

Figure 7.15: Hexagonal Tile Assembly Model—0-1 Knapsack Solution—True Negative. Failed by first verification.

configuration is because the pink verification tiles known as the "Greater Than Or Equal To Zero" verification tiles (Figure 7.8 in Chapter 7) have assembled such that only one empty position in the configuration exists with three neighbours, however there is no tile defined in the set $\mathbb{T}$ that will fit in this position. This consequence is a desired feature of these verification tiles. The tiles are enforcing the value constraint of the 0-1 Knapsack problem definition. In this case, the value 2 is not greater than or equal to the value $V = 17$ therefore the true negative is the correct output of this configuration.

Fail by second verification

The second way a checkmark tile may fail to assemble in a final configuration is depicted by Figure 7.16. The orange tiles at the top of the final configuration assemble only if the first verification (on the right hand side) assembles with a 0 in the MSB, as is the case with this image. Recall, the 0 in the MSB of the greater than or equal verification tiles imply a successful validation of the value constraint.

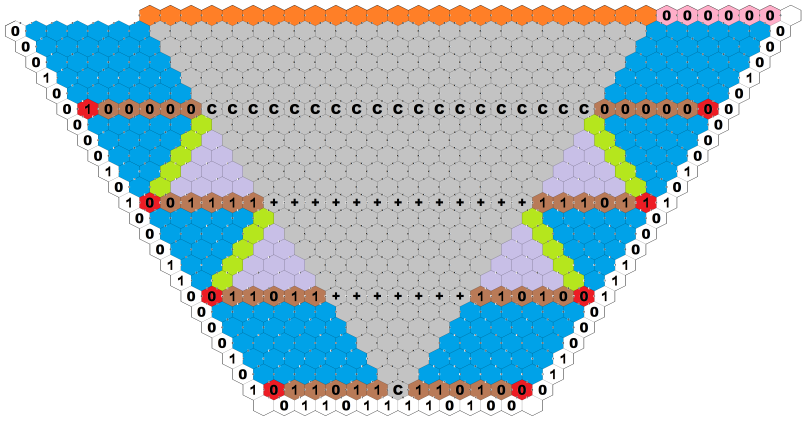

Figure 7.16: Hexagonal Tile Assembly Model—0-1 Knapsack Solution—True Negative. Failed by second verification.

Where the verification fails here then, is the left hand side. In particular, the tiles that are designed to possibly assemble in those empty positions are the "Less Than Zero" verification tiles defined by Figure 7.11. However, in this final configuration, the "Less Than Zero" tiles will not be able to assemble, because the the MSB of the aggregate weight value, $-W + \sum \boldsymbol{w}_i \boldsymbol{x}_i$, is 0. Recall when a two's complement integer has a 0 MSB, then that value is positive. The initial value of the left hand side, encoded by the starting configuration, is $-W$. Therefore the sum of the weights of chosen items in the knapsack exceeds W; the maximum constraint of weight. Hence the "Less Than Zero" tiles should not be allowed to assemble when the MSB is 0, as is the case. The lack of tiles in the empty positions in turn assures the checkmark tile is not present in the final configurations that do not contain these verification tiles.

Fail by variables not matching

The third example of a true negative is shown in Figure 7.17. This figure shows a final configuration where a section of the grey, middle verification tiles cannot

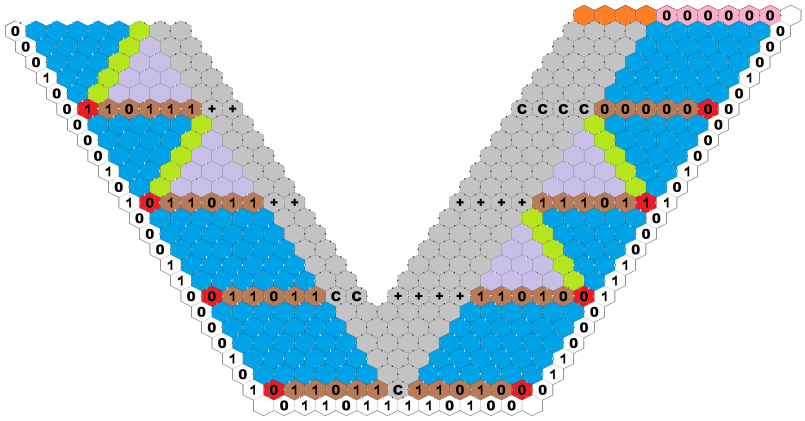

Figure 7.17: Hexagonal Tile Assembly Model—0-1 Knapsack Solution—True Negative. Failed by variables not matching and lateral synchronization verification.

assemble, in addition to the checkmark tile. The empty position with greater than three neighbours is in between the 'C' and '+' tiles on the same line. By design, a 'C' and '+' tile on the same line results in a negative result for this assembly, as is the case here. The purpose of those 'C' and '+' tiles is to only allow the grey verification tiles to fully assemble in all central positions if the two red non-deterministic tiles on opposite ends of the same row have the same function, copy or addition respectively. This translates to the 0-1 Knapsack problem as ensuring that while adding an objects weight to the knapsack, the value is also added, and vice versa. As a consequence of this verification, the checkmark tile will not assemble as part of a final configuration if there is not an exact match of copy and addition operations chosen for each object.

It is interesting to note a couple of points before moving on from the example in Figure 7.17. Notice the tiles have assembled to the top row even though there are positions unfilled in the seventh row and each row after that. The point here is that tiles may assemble along the left and right side without needing to assemble each

row below it, however a row must not contain an empty position in order for the row above it to fill all of its empty positions. Unlike the Wang et al.'s solution, the empty position in the relevant row cannot have a tile attach adjacent and above it. The hexagonal assembly cannot build out from the empty positions because each other position beyond row 7 in Figure 7.17 has only at most two neighbours (recall that for the 6-TAM solution, we set $\tau = 3$). The fact that no tiles can legally assemble adjacent to the hole in row 7 won't change because this figure illustrates a final configuration, meaning there are no more tiles that can legally attach to this configuration at it's current state.

Also of interest, both the left hand side (object weights) and the right hand side (object values) in this figure have been chosen such that the total sums was smaller than their respective constraints. This fact alone would normally imply the verification tiles would pass validation, however the design of this system is such that the grey "Lateral Synchronization" verification tiles take precedence in validating the result of this assembly. Specifically, the top row verification tiles are not capable of assembling a complete row, regardless of values chosen, if there is a hole at any point in the assembly. This fact is by consequence of the grey verification tiles being assembled prior to the top row and the fact that the top row cannot complete unless all rows below it are complete.

Fail by verification tiles being chosen non-deterministically

The final scenario that can cause a true negative result is illustrated in Figure 7.18. This scenario is slightly different than the previous three examples because those ex-

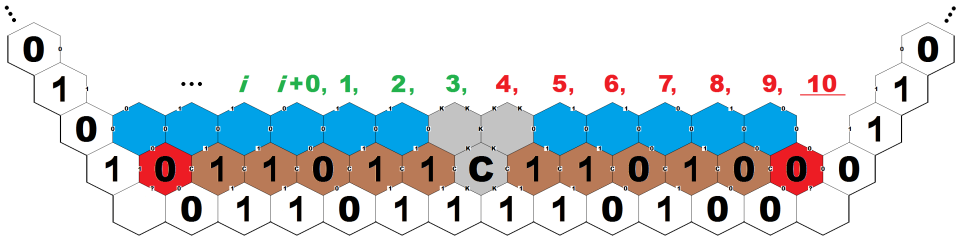

Figure 7.18: Hexagonal Tile Assembly Model—0-1 Knapsack Solution—True Negative. Failed by lateral synchronization placed non-deterministically at the "wrong time".

amples resulted in designed features of the tiles in this set to determine a true negative, whereas the result in this scenario is a true negative produced by a coincidental circumstance, as a consequence of the non-deterministic style that the tile assemble model chooses tiles to assemble at each step. In Figure 7.18, we've illustrated only the bottom four rows of a non-final configuration. This will be enough of the image and far enough along in the assembly process to explain how the final configuration will be negative. Note the indexing in the second row. The numbers show the column index, relative to column i. In this example, we will also interpret the column indices to be counting the order the tiles have assembled using the assumption they are assembling exclusively from left-to-right. This specific ordering is not guaranteed for the general case, but it's a convenient assumption for making the argument that this scenario exists and may occur in some situations. The place where the colour of the indices changes from green to red simply is emphasizing the step in the assembly where the failure occurs. We will explain further how this causes a negative result. First, recall the way bidirectional copy tiles work. In addition to the vertical direction, the horizontal direction is transferred laterally from the tiles neighbour to its

opposite adjacent position on the same row. However, if the opposite position is not empty, then the bidirectional tile is able to assembly there if and only if it matches the abutting edges on both sides.

This scenario is one such that the a bidirectional tile may not assemble in an empty position that would normally be available for such a tile. In the example we've illustrated, the non-deterministic tile on both the left and right sides is the copy operation. Then, starting from column i, let us assume the tiles in this row are assembling in the order from left-to-right and the row below has already been completely filled in all positions. Then notice at column 3 that the verification tile assembles with the West edge being a 0. Next, the tile in column 4 assembles, but note that since column 5 would still be empty, then there are two tiles that can legally fit in position 4. Namely, the two verification tiles with East edges equal to 0 and 1. We can see that this tile was chosen non-deterministically, however it is the wrong choice. This is shown in column 10 because no tile can assemble there now. Therefore, we've shown a situation where the non-deterministic ordering of tiles can create a choice in the tiles on arbitrary rows and cause negative results, regardless of which variables are chosen by the red non-deterministic choice tiles.

7.2.3 False Negative

False negative is a case that happens to behave exactly like the true negative case. In short, false negatives may occur in the same ways that we've described previously for true negatives. In the explanations for the true negative cases, you will notice that we show those scenarios give a negative result, but it is unspecified if the result

is true or false negative. Conveniently, it is an unnecessary distinction to the tile assembly model techniques for solving NP-Complete problems. This is because the massively parallel, non-deterministic iterations of the assembly makes it so the result is decided only by the presence of a checkmark tile denoting a true negative result.

7.2.4 False Positive

It's important to note that there does not exist a False Positive category. Great care must be taken while defining massively parallel, non-deterministic tile assembly models of any degree, to define tiles such that no final configurations would be categorized under the erroneous "False Positive" classification. This is due to the way the 6-TAM definitions ensure programmatic correctness from the massively parallel, non-deterministic technique employed to produce final configurations en masse and filter correct solutions to the problem attempting to be solved. Experimentally speaking, if there exists an observable positive final configuration in the complete set of produced final configurations, then it's concluded that the starting configuration determined a representation of inputs to a "yes" instance. Introducing a False Positive outcome into the tile set definition would cause a fallacy in the TAM strategy for solving NP-Complete decision problems.

To see why TAM decision problems require that False Positives cannot exist, consider the scenario where there exists two categorizations of positive final configurations, namely one true and one false. Then, there would no longer be the guarantee that the result of this ill-defined tile set would be well formed, in a logical sense. To see why, consider what occurs with regards to the following contradictory outcome.

Suppose that there is a starting configuration with inputs representing a "no" instance of some problem, however the final configuration contains a checkmark tile. Therefore, the existence of the checkmark tile was not decided on the implication that the starting configuration began with an encoding of some "yes" instance. It's easy to see this scenario would be troublesome because then there would be no logical guarantee that "yes" and "no" instances indicate the results of well defined tiles. This fallacy is realized because a positive output, which supposedly indicates a "yes" instance, could equally be either true or false, i.e., the starting configuration could actually be either a "yes" instance or a "no" instance. Since evidently "yes" instances should not be recognized as "no" instances, and vice versa, this fallacy needs to be avoided. Hence, tiles are defined such that False Positives are not possible.

Chapter 8

Summary

The hexagon tile assembly model has been shown to be a robust and useful extension of the existing tile assembly model research. The current literature was able to make significant headway in the research of solving computational problems using DNA self-assembling biomolecular tiles. From elementary arithmetic to NP-Complete problems, the square tile assembly model was able to be an essential piece of building a solution. As in any research, as the field expanded and more discoveries where uncovered, so too were more questions exposed. In this thesis, we first showed the background for the tile assembly model in Chapter 2. In Chapter 3, we gave examples how the model is used in existing literature. After that, we proceeded to demonstrate where the existing methodologies of using square DNA tiles had limitations for solving problems whose starting configuration would use a U-shape. Therefore, we defined the hexagon tile assembly model systems to compare with existing material, showing in Chapter 6 that it is possible to perform many of the same computations with hexagons as are already known with squares. In Chapter 5, we was able to highlight a

result where the square tiles were insufficient to solve U-shape starting configurations, however the hexagon tiles excelled in this scenario.

Chapter 5 motivated the requirement for an improvement on square tile assembly models. A case study was examined where a solution for the well known NP-Complete problem 0-1 Knapsack was attempted using square tiles, but a flaw was exposed. Namely, the U-shape starting configuration with square tiles is not feasible due to the non-deterministic self-assembly property of the tile assembly model. The tiles in that scenario could assemble out of order and thereby yield results that are non-valid, since the order of assembly is an important feature that must be carefully considered while designing the system. In the first half of Chapter 7, we showed that hexagon tile assembly model can be used to solve the 0-1 Knapsack problem. We defined hexagon tiles to solve the 0-1 Knapsack problem and showed that they don't suffer the same issue, regarding the starting configuration, as the analogous square tile system. Next, in the second half of Chapter 7 we showed the three specific outcomes of a given instance of the self-assembly process for the 0-1 Knapsack solution using hexagon tiles. We explained in this section how there are multiple final configurations expected as output to the self-assembly process and we showed how to interpret the results as a whole when taking all final assembly instances together.

Throughout this thesis, we made comparisons between the existing square tile assembly model and the novel introduction of the hexagon tile assembly model. Particularly in Chapter 6, we showed how for many previously solved problems, using square tiles, an analogous hexagon tile solution was also available. Moreover, as shown in Chapter 5, the square 0-1 Knapsack solution had a flaw that was remedied

by the conversion to hexagon tiles. It is therefore natural to discuss if there is a generalization on how or when hexagon tiles may improve on a problem's solution. In other words, when should square tiles be used and when should hexagons be used? Is there any reason for not using hexagons? After all, the complexity of hexagons was simpler than squares in all the examples shown. Having additional edges exposed was a recurring theme that was a convenience for hexagons. Conversely, squares would require additional tiles and "intermediary edge states" such that those tiles' edges might be defined in a way to determine the state of a neighbour. That said, the modified value had no semantic effect on the meaning of that tile. Logically, one can think of such tiles as treating the modified value as "this value is not mine. I'm just passing it to someone else". This never happened in hexagon tiles and the logical meaning of each tile was never resolved by use of a notation to indicate various states of the same tile. Each tile in 6-TAM had its own state and mapped to a single, logical step in the computation being solved by the system in an intuitive way. This effectively limits the required number of alternate edges and alternate tiles required for a subsystem to function. Therefore, we showed a major improvement over square TAM systems was the hexagon's ability to be minimal in complexity because given the choice between squares and hexagons, hexagons lead to fewer tile and edge definitions.

Appendix A

Wang et al. Tile Assembly Model Solution for the 0-1 Knapsack Problem

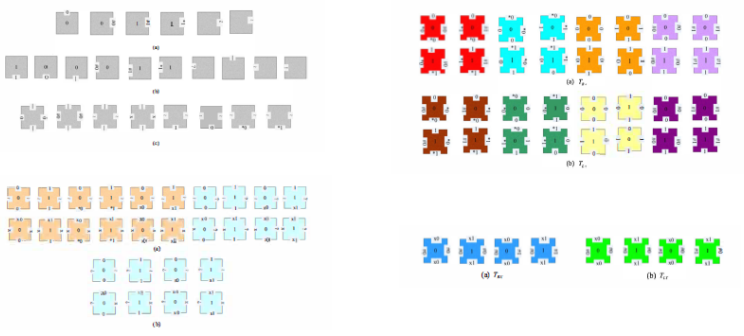

Figure A.1: Wang et al.'s 4-TAM Solution for the 0-1 Knapsack Problem

Appendix B

0-1 Knapsack Hexagonal Tile Assembly Solution Formulas

B.1 Starting Tiles

B.1.1 Tile Definitions

$\forall a \in \{0, 1\}$,

a. $\boldsymbol{s}_1(a) = \langle \text{BLANK}, \text{BLANK}, a, \text{BLANK}, \text{NULL}, \text{NULL} \rangle$,

b. $\boldsymbol{s}_2(a) = \langle a, \text{BLANK}, \text{BLANK}, \text{NULL}, \text{NULL}, \text{BLANK} \rangle$,

c. $\boldsymbol{s}_3 = \langle 0, ?, \text{BLANK}, \text{NULL}, \text{NULL}, \text{BLANK} \rangle$,

d. $\boldsymbol{s}_4 = \langle \text{NULL}, \text{NULL}, D, 0, \text{BLANK}, \text{NULL} \rangle$,

e. $\boldsymbol{s}_5 = \langle \text{BLANK}, ?, \text{BLANK}, \text{NULL}, \text{NULL}, \text{NULL} \rangle$,

f. $\boldsymbol{s}_6 = \langle \text{BLANK}, K, 1, \text{BLANK}, \text{NULL}, \text{NULL} \rangle$, and

g. $\boldsymbol{s}_7 = \langle G, \text{NULL}, \text{NULL}, \text{NULL}, \text{NULL}, \text{BLANK} \rangle$.

Let Γ be the set of all starting tiles. The set Γ contains all $\boldsymbol{s}_i(a)$, $\forall i \in \{1, 2, \ldots, 7\}$ and their mirror images $\overline{\boldsymbol{s}_i}(a)$. The starting configuration for 6-TAM solution of the 0-1 Knapsack Problem is denoted by $S : \mathbb{Z}^2 \mapsto \Gamma$.

$$\Gamma = [\bigcup_{\substack{a \in \{0,1\} \\ i \in \{1,2\}}} \boldsymbol{s}_i(a) \cup \overline{\boldsymbol{s}_i}(a)] \cup [\bigcup_{i=3}^{7} \boldsymbol{s}_i \cup \overline{\boldsymbol{s}_i}] \tag{B.1}$$

B.1.2 Labels

$\forall a \in \{0, 1\}$,

- $v(\boldsymbol{a})) = v(\overline{\boldsymbol{s}_1}(a)) = a$,
- $v(\boldsymbol{s}_2(a)) = v(\overline{\boldsymbol{s}_2}(a)) = a$,
- $v(\boldsymbol{s}_3) = v(\overline{\boldsymbol{s}_3}) = 0$,
- $v(\boldsymbol{s}_4) = v(\overline{\boldsymbol{s}_4}) = 0$,
- $v(\boldsymbol{s}_5) = \text{BLANK}$,
- $v(\boldsymbol{s}_6) = \text{BLANK}$, and
- $v(\boldsymbol{s}_7) = \text{BLANK}$.

B.2 Copy Tiles

B.2.1 Tile Definitions

Unidirectional

$\forall a \in \{0, 1\}$,

a. $\boldsymbol{t}_1(a) = \langle \text{BLANK}, \text{BLANK}, a, \text{BLANK}, \text{BLANK}, a \rangle$.

$$T_1 = [\bigcup_{a \in \{0,1\}} \boldsymbol{t}_1(a) \cup \overline{\boldsymbol{t}_1}(a)] \tag{B.2}$$

Bidirectional

$\forall a, b \in \{0, 1\}$,

a. $\boldsymbol{t}_2(a, b) = \langle a, \text{BLANK}, b, a, \text{BLANK}, b \rangle$.

$$T_2 = [\bigcup_{a,b \in \{0,1\}} \boldsymbol{t}_2(a, b) \cup \overline{\boldsymbol{t}_2}(a, b)] \tag{B.3}$$

B.2.2 Labels

Unidirectional

$\forall a \in \{0, 1\}$,

- $v(\boldsymbol{t}_1(a)) = a$.

Bidirectional

$\forall a, b \in \{0, 1\}$,

- $v(\boldsymbol{t}_2(a, b)) = a$.

B.3 Bidirectional Copy Tiles with Verification Inputs

B.3.1 Tile Definitions

$\forall a \in \{0, 1\}$ and $\forall \alpha \in \{+, \mathrm{C}\}$,

a. $\boldsymbol{t}_3(a, \alpha) = \langle \alpha, \mathrm{BLANK}, a, \alpha, \mathrm{BLANK}, a \rangle$.

$$T_3 = [\bigcup_{\substack{a \in \{0,1\} \\ \alpha \in \{+,\mathrm{C}\}}} \boldsymbol{t}_3(a, \alpha) \cup \overline{\boldsymbol{t}_3}(a, \alpha)] \tag{B.4}$$

B.3.2 Labels

$\forall a \in \{0, 1\}$ and $\forall \alpha \in \{+, \mathrm{C}\}$,

- $v(\boldsymbol{t}_3(a, \alpha)) = a$.

B.4 Addition Tiles

B.4.1 Tile Definitions

$\forall a, b, c \in \{0, 1\}$,

a. $\boldsymbol{t}_4(a,b,c) = \langle \text{BLANK}, (a+b+c) > 1, (a+b+c) \mod 2, b, c, a \rangle$.

$$T_4 = [\bigcup_{a,b,c \in \{0,1\}} \boldsymbol{t}_4(a,b,c) \cup \overline{\boldsymbol{t}_4}(a,b,c)] \tag{B.5}$$

B.4.2 Labels

$\forall a, b, c \in \{0,1\}$,

- $v(\boldsymbol{t}_4(a,b,c)) = (a+b+c) \mod 2$.

B.5 Non-deterministic Tiles

B.5.1 Tile Definitions

$\forall a, b \in \{0,1\}$,

a. $\boldsymbol{t}_{5_1}(a,b) = \langle +, (a+b) > 1, (a+b) \mod 2, b, ?, a \rangle$,

b. $\boldsymbol{t}_{5_2}(a,b) = \langle \text{C}, \text{BLANK}, a, b, ?, a \rangle$.

$$T_5 = [\bigcup_{\substack{a,b \in \{0,1\} \\ i \in \{1,2\}}} \boldsymbol{t}_{5_i}(a,b) \cup \overline{\boldsymbol{t}_{5_i}}(a,b)] \tag{B.6}$$

B.5.2 Labels

$\forall a, b \in \{0,1\}$,

- $v(\boldsymbol{t}_{5_1}(a,b)) = a+b \mod 2$, and
- $v(\boldsymbol{t}_{5_2}(a,b)) = a$.

B.6 Lateral Synchronization

B.6.1 Tile Definitions

$\forall a \in \{0,1\}$ and $\forall \alpha \in \{+, \mathrm{C}\}$,

a. $\boldsymbol{t}_{6_1}(\alpha) = \langle \alpha, \mathrm{K}, \mathrm{K}, \alpha, \mathrm{BLANK}, \mathrm{K} \rangle$,

b. $\boldsymbol{t}_{6_2}(\alpha) = \langle \alpha, \mathrm{K}, \mathrm{K}, \alpha, \mathrm{K}, \mathrm{K} \rangle$,

c. $\boldsymbol{t}_{6_3}(a) = \langle \mathrm{C}, \mathrm{K}, \mathrm{K}, \mathrm{C}, a, \mathrm{K} \rangle$,

d. $\boldsymbol{t}_{6_4}(a) = \langle \mathrm{K}, \mathrm{K}, \mathrm{K}, a, \mathrm{BLANK}, \mathrm{K} \rangle$,

e. $\boldsymbol{t}_{6_5} = \langle +, \mathrm{K}, \mathrm{K}, +, 0, \mathrm{K} \rangle$,

f. $\boldsymbol{t}_{6_6} = \langle \mathrm{K}, \mathrm{K}, \mathrm{K}, \mathrm{BLANK}, \mathrm{BLANK}, \mathrm{K} \rangle$, and

g. $\boldsymbol{t}_{6_7} = \langle \mathrm{K}, \mathrm{K}, \mathrm{K}, \mathrm{K}, \mathrm{K}, \mathrm{K} \rangle$.

$$T_6 = [\bigcup_{\substack{\alpha \in \{(+),(C)\} \\ i \in \{1,2\}}} \boldsymbol{t}_{6_i}(\alpha) \cup \overline{\boldsymbol{t}_{6_i}}(\alpha)] \cup [\bigcup_{\substack{a \in \{0,1\} \\ i \in \{3,4\}}} \boldsymbol{t}_{6_i}(a) \cup \overline{\boldsymbol{t}_{6_i}}(a)] \cup [\bigcup_{i \in \{5,6\}} \boldsymbol{t}_{6_i} \cup \overline{\boldsymbol{t}_{6_i}}] \cup \{\boldsymbol{t}_{6_7}\} \quad \text{(B.7)}$$

B.6.2 Labels

$\forall a \in \{0,1\}$ and $\forall \alpha \in \{+, \mathrm{C}\}$,

- $v(\boldsymbol{t}_{6_1}(\alpha)) = \alpha$,
- $v(\boldsymbol{t}_{6_2}(\alpha)) = \alpha$,
- $v(\boldsymbol{t}_{6_3}(a)) = \mathrm{C}$,

- $v(\boldsymbol{t}_{6_4}(a)) = \mathrm{K}$,
- $v(\boldsymbol{t}_{6_5}) = +$,
- $v(\boldsymbol{t}_{6_6}) = \mathrm{K}$, and
- $v(\boldsymbol{t}_{6_7}) = \mathrm{K}$.

B.7 Greater Than Or Equal To Zero

B.7.1 Tile Definitions

$\forall \alpha \in \{\mathrm{E}, \mathrm{F}\}$,

a. $\boldsymbol{t}_{7_1} = \langle \mathrm{E}, \mathrm{NULL}, \mathrm{NULL}, \alpha, \mathrm{BLANK}, 0 \rangle$ and

b. $\boldsymbol{t}_{7_2} = \langle \mathrm{F}, \mathrm{NULL}, \mathrm{NULL}, \alpha, \mathrm{BLANK}, 1 \rangle$.

$$T_7 = \bigcup_{\alpha \in \{\mathrm{E},\mathrm{F}\}} \{\boldsymbol{t}_{7_1}(\alpha), \boldsymbol{t}_{7_2}(\alpha)\} \tag{B.8}$$

B.7.2 Labels

- $v(\boldsymbol{t}_{7_1}) = 0$ and
- $v(\boldsymbol{t}_{7_2}) = 1$.

B.8 Top Row Verification Transmission

B.8.1 Tile Definitions

$\forall a \in \{0, 1\}$,

a. $\boldsymbol{t}_{8_1}(a) = \langle \mathrm{E}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{E}, a, \mathrm{K} \rangle$,

b. $\boldsymbol{t}_{8_2} = \langle \mathrm{E}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{E}, \mathrm{BLANK}, \mathrm{K} \rangle$,

c. $\boldsymbol{t}_{8_3} = \langle \mathrm{E}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{E}, \mathrm{K}, 0 \rangle$, and

d. $\boldsymbol{t}_{8_4} = \langle \mathrm{E}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{E}, \mathrm{K}, \mathrm{K} \rangle$.

$$T_8 = \{ \bigcup_{a \in \{0,1\}} \boldsymbol{t}_{8_1}(a) \} \cup \{ \boldsymbol{t}_{8_2}, \boldsymbol{t}_{8_3}, \boldsymbol{t}_{8_4}, \overline{\boldsymbol{t}_{8_2}} \} \tag{B.9}$$

B.8.2 Labels

$\forall a \in \{0, 1\}$,

- $v(\boldsymbol{t}_{8_1}(a)) = \mathrm{K}$,
- $v(\boldsymbol{t}_{8_2}) = \overline{\boldsymbol{t}_{8_2}} = \mathrm{K}$,
- $v(\boldsymbol{t}_{8_3}) = \mathrm{K}$, and
- $v(\boldsymbol{t}_{8_4}) = \mathrm{K}$.

B.9 Less Than Zero

B.9.1 Tile Definitions

a. $\boldsymbol{t}_{9_1} = \langle \mathrm{L}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{L}, 0, \mathrm{BLANK} \rangle$,

b. $\boldsymbol{t}_{9_2} = \langle \mathrm{L}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{L}, 1, \mathrm{BLANK} \rangle$, and

c. $\boldsymbol{t}_{9_3} = \langle \mathrm{L}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{E}, 1, \mathrm{BLANK} \rangle$.

$$T_9 = \{\boldsymbol{t}_{9_1}, \boldsymbol{t}_{9_2}, \boldsymbol{t}_{9_3}\} \tag{B.10}$$

B.9.2 Labels

- $v(\boldsymbol{t}_{9_1}) = 0$,
- $v(\boldsymbol{t}_{9_2}) = 1$, and
- $v(\boldsymbol{t}_{9_3}) = 1$.

B.10 Checkmark

B.10.1 Tile Definitions

$\forall a \in \{0, 1\}$

a. $\boldsymbol{t}_{10}(a) = \langle \mathrm{NULL}, \mathrm{NULL}, \mathrm{NULL}, \mathrm{L}, a, \mathrm{D} \rangle$.

$$T_{10} = \bigcup_{a \in \{0,1\}} \boldsymbol{t}_{10}(a) \tag{B.11}$$

B.10.2 Labels

$\forall a \in \{0, 1\}$

- $v(\boldsymbol{t}_{10}(a)) = \checkmark$.

B.10.3 Final Equation

Finally, the following equation brings all the subsets together into one set.

$$\mathbb{T} = \bigcup_{i=1}^{10} T_i \tag{B.12}$$

Appendix C

Long Multiplication

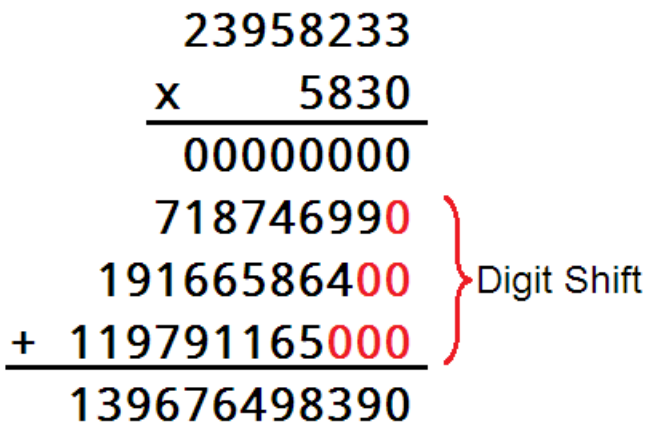

Figure C.1: Long Multiplication example, from Wikipedia [18]. Note the digit shift and addition is part of the algorithm.

Bibliography

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, November 1994.

[2] M. Amos. *Theoretical and Experimental DNA Computation*. Natural Computing Series. Springer-Verlag Berlin Heidelberg, 2005.

[3] Y. Brun. Arithmetic computation in the tile assembly model: Addition and multiplication. *Theoretical Computer Science*, 378(1):17–31, June 2007.

[4] Y. Brun. Solving NP-complete problems in the tile assembly model. *Theoretical Computer Science*, 395(1):31–46, April 2008.

[5] Y. Brun. Solving satisfiability in the tile assembly model with a constant-size tileset. *Journal of Algorithms*, 63(4):151–166, October 2008.

[6] M. Gardner. The fantastic combinations of John Conway's new solitaire game "Life". *Scientific American*, 223:120–123, 1970.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[8] J. Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1–3):3–33, 2005.

[9] L. Kari, S. Seki, and Z. Xu. Triangular and hexagonal tile self-assembly systems. In *Computation, Physics and Beyond*, pages 357–375. Springer, 2012.

[10] E. Katz. *Biomolecular Information Processing: From Logic Systems to Smart Sensors and Actuators*. Wiley, 2013.

[11] M. Linial and N. Linial. On the potential of molecular computing. *Science-AAAS-Weekly Paper Edition*, 268(5210):481–484, April 1995.

[12] P. W. K. Rothemund. Scaffolded DNA origami: From generalized multicrossovers to polygonal networks. In G. Rozenberg, T. Bäck, J. N. Kok, H. P. Spaink, A. E. Eiben, J. Chen, and N. Jonoska, editors, *Nanotechnology: Science and Computation*, pages 3–21. Springer Berlin Heidelberg, Berlin, 2006.

[13] P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *STOC'00: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 459–468, Portland, Oregon, United States, May 2000. ACM.

[14] P. W. K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol*, 2(12):e424, 2004.

[15] H. Wang. Proving theorems by pattern recognition. II. *Bell System Technical Journal*, 40(1):1–41, January 1961.

[16] Y. Wang, W. Lu, X. Zhang, and G. Cui. DNA tile assembly model for 0–1 knapsack problem. *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, pages 180–184, 2010.

[17] Wikipedia. Nucleic acid thermodynamics — Wikipedia, the free encyclopedia, 2014. URL `http://en.wikipedia.org/w/index.php?title=Nucleic_acid_thermodynamics&oldid=629552703`. [Online; accessed 4-January-2015].

[18] Wikipedia. Multiplication algorithm — Wikipedia, the free encyclopedia, 2014. URL `http://en.wikipedia.org/wiki/Multiplication_algorithm#Long_multiplication`. [Online; accessed 31-October-2014].

[19] E. Winfree. On the computational power of DNA annealing and ligation. In *DNA Based Computers, volume 27 of Proceedings of the DIMACS Workshop*, pages 199–221. American Mathematical Society, 1995.

[20] X. Zhang, Y. Wang, Z. Chen, J. Xu, and G. Cui. Arithmetic computation using self-assembly of DNA tiles: Subtraction and division. *Progress in Natural Science*, 19(3):377–388, March 2009.