Tile Self-Assembly

Sarah Allen Kristy Gardner

May 11, 2015

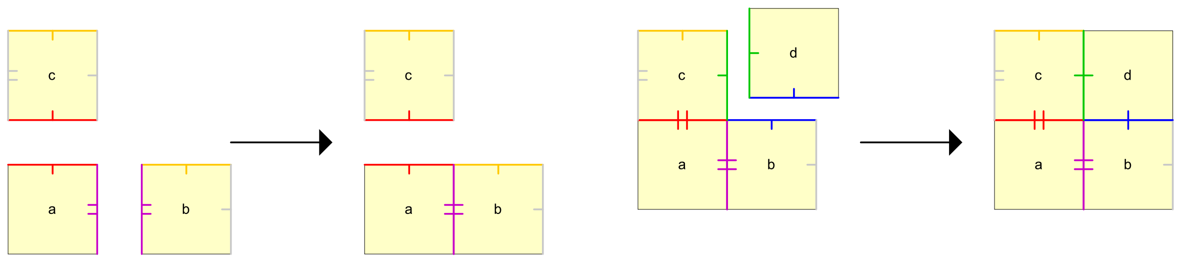
1 Introduction

The goal of DNA computing is to perform computations using biological components rather than traditional electronic circuits. Molecules can combine to form larger structures if they contain complementary strands of DNA. These structures, called *assemblies*, can be interpreted as the output of a program. The inherently parallel nature of chemical reactions enables this model of computation to simultaneously explore all paths in a nondeterministic computation tree [Adl94].

In this project, we explore an abstraction of DNA computing called the *Tile Assembly Model* (TAM) [Win98], which is known to be Turing complete. The TAM models molecules as a set of square tiles that self-assemble to tile the plane in a square lattice. A program consists of a finite set of tile types (it is assumed that there is an unlimited number of tiles of each type available), which may be placed in any location on the lattice, but not rotated. Once a tile has been placed in the lattice, it cannot be moved or removed. The program begins with a single seed tile to which other tiles can bond. The program continues running until it is not possible to add any more tiles to the assembly. The resulting assembly is called a *terminal assembly*.

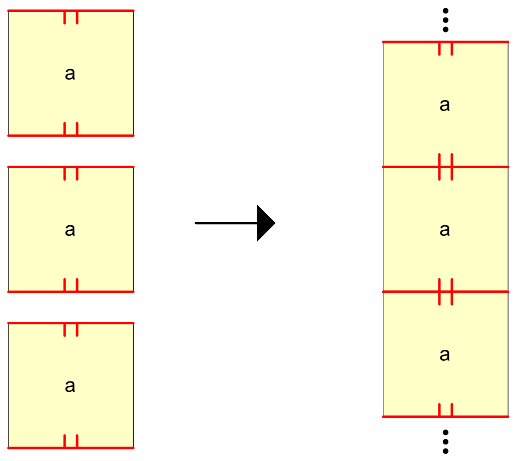
Two tiles can bond depending on their types. A tile's type is determined by the *glue* on each of its four sides. Each glue has a particular color and integral strength. A tile may bond with another tile if their adjacent sides have glues of the same color and strength at least 2. For example, in Figure 1a, tile *b* can attach to tile *a* because the glue that they share has strength 2. Tile *c* cannot attach because its bottom glue only has strength 1. In addition to two individual tiles bonding, a tile can bond to an existing assembly of multiple tiles if the sum of glue strengths of all matching glues is at least 2 (see Figure 1b). Mismatched glues do not prevent bonding; they simply do not affect the formation of new bonds. Glues may also be null; a null glue cannot bind two tiles regardless of the strength (a null glue is equivalent to a glue of strength 0).

One way of creating a correct tile set is to simply create a specific set of glues for each position in the desired terminal assembly. However, this requires an extremely large number of different tile types and glues, which can be difficult to design in practice. Hence an important concern in designing tile sets is using a

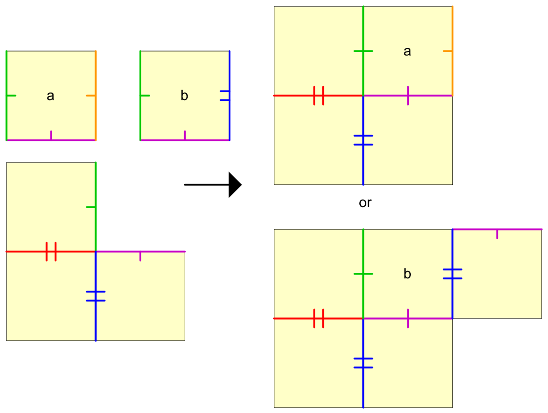


(a) Tile *b* can attach to tile *a*, but tile *c* cannot. (b) Tile *d* can attach because it has two strength-1 glues.

Figure 1: Basic rules for attachment



(a) This produces an infinite chain of tile a.



(b) Either a or b can attach.

Figure 2: Some tile sets with ambiguous terminal configurations

small number of distinct types of tiles and glues. By repeating glues and tile types, we reduce the complexity of the system, but it is no longer obvious that a given tile set produces a unique terminal assembly. For example, the tile set depicted in Figure 2a can form an arbitrarily long chain. Furthermore, since the same glue can occur on multiple tiles, there may be multiple options for which tile attaches to a particular location in the assembly. For example, in Figure 2b, either tile a or tile b can attach to the empty space, producing two different assemblies.

Given that the purpose of a tile set is to produce a *unique* terminal assembly, it is important to find some way of verifying that a tile set does in fact achieve this. We consider a set of tiles to be *correct* if it produces a unique terminal assembly. Our goal in this project is to develop a program that verifies the correctness of a tile set. In particular, since tile self-assembly is inherently nondeterministic, our goal is to demonstrate that for a given tile set and seed, all possible execution paths (sequences of sub-assemblies) yield the same terminal assembly. We make three contributions towards this goal:

- We *augment the imperative programming language* to include constructs that represent the components of the tile assembly system, and we *define the semantics* for these new elements of the language.
- We use this augmented language to *write a program for tile assembly verification*. The program is very general and does not assume a particular tile set or glue set. We *determine the semantics* of the helper functions that we define and of the program as a whole.
- We *verify the correctness* of the tile set for an $m \times n$ rectangle using the semantics of our program.

2 Model

We augment the imperative programming language introduced in class to include types related to the tile assembly problem. The key component in the tile assembly problem is a *tile*. A tile is a 1×1 square defined by the *glue* on each of its sides. Tiles may not rotate. Each glue has a color and a strength, and two tiles can join if the glues on their adjacent sides are of the same color and have strength at least 2. A tile may also join a substructure if the total glue strength between the tile and matching glues on adjacent edges of the substructure is at least 2. The tiles assemble in a grid, hence we also need a *location* that indicates a tile's position in the grid. Once a tile has been placed at a particular location, it cannot be moved, removed, or replaced. An assembly of one or more tiles is called a *structure*. We represent a structure as an array of tiles indexed by their location. For example, $\mathtt{S}(0, 0)$ indicates the tile located at the origin.

We add new types to the language for tiles, glues, and locations. A location is simply an ordered pair of x and y coordinates:

$$\texttt{Location} := (\texttt{E}_1, \texttt{E}_2)_\ell \qquad \texttt{E}_1, \texttt{E}_2 \in \texttt{Expr}.$$

At each location, there may be a tile, hence one way of representing a tile is by indexing into a structure at a particular location; a tile can be an array access $\texttt{S}(x, y)$. It is possible that there is no tile at the location (x, y), in which case the tile is represented by `noTile`. If there is a tile, it is represented as an ordered 4-tuple of glues (n, s, e, w), where the first element is the glue on the top edge of the tile, the second is the glue on the bottom edge of the tile, the third element is the glue on the right edge of the tile, and the fourth element is the glue on the left edge of the tile. Hence our syntax for tiles is:

$$\texttt{Tile} := (n, s, e, w) \mid \texttt{noTile} \mid \texttt{S}(x, y) \qquad n, s, e, w \in \texttt{Glue},\ x, y \in \mathbb{Z}.$$

A glue is an ordered pair of integers, where the first integer indicates the glue's color, and the second integer indicates the glue's strength. The glue color may be null, in which case the glue can never bind two tiles together regardless of the strength. For ease of usage, we also allow a glue to be indicated by indexing into a structure using a location and an integer representing the edge of the tile at that location. The syntax for glues is thus:

$$\texttt{Glue} := (\texttt{E}_1, \texttt{E}_2)_g \mid (\texttt{nullGlue}, \texttt{E}_2)_g \mid \texttt{S}(\texttt{E}_3, \texttt{E}_4, \texttt{E}_5) \qquad \texttt{E}_i \in \texttt{Expr},\ [\![\texttt{E}_5]\!] \in \{0, 1, 2, 3\},$$

Finally, we include constructs for sets of tiles and sets of glues:

$$\texttt{TileSet} := \emptyset \mid \{\texttt{T}\} \mid \{\texttt{T}\} \cup \texttt{TS}$$

$$\texttt{GlueSet} := \emptyset \mid \{\texttt{G}\} \mid \{\texttt{G}\} \cup \texttt{GS}.$$

3 Semantics

In this section, we define the semantics for the components that we added to the language. Tiles, glues, and locations can all be specified syntactically using expressions. Since the semantic meaning of an expression depends on the state, the semantic meanings of tiles, glues, and locations also depend on the state. Hence the semantics for each of these constructs is a function that maps from states to some other space.

A location simply maps to a coordinate in $\mathbb{Z}^2$:

$$\frac{[\![\cdot]\!] :\ \texttt{Location} \to \texttt{St} \to \mathbb{Z}^2}{[\![(\texttt{E}_1, \texttt{E}_2)_l]\!]\nu = ([\![\texttt{E}_1]\!]\nu, [\![\texttt{E}_2]\!]\nu)}$$

A glue also maps to a coordinate in $\mathbb{Z}^2$, except that the first coordinate may also be `nullGlue`:

$$\underline{[\![\cdot]\!] :\ \texttt{Glue} \to \texttt{St} \to (\{\texttt{nullGlue}\} \cup \mathbb{Z}) \times \mathbb{Z}}$$

$$[\![(\texttt{nullGlue}, \texttt{E}_2)_g]\!]\nu = (\texttt{nullGlue}, 0)$$

$$[\![(\texttt{E}_1, \texttt{E}_2)_g]\!]\nu = ([\![\texttt{E}_1]\!]\nu, [\![\texttt{E}_2]\!]\nu)$$

$$[\![\texttt{S}(\texttt{E}_3, \texttt{E}_4, \texttt{E}_5)]\!]\nu = \begin{cases} \texttt{nullGlue} & \text{if } \texttt{S}(\texttt{E}_3, \texttt{E}_4) = \texttt{noTile} \\ \nu(\texttt{S})([\![\texttt{E}_3]\!]\nu, [\![\texttt{E}_4]\!]\nu, [\![\texttt{E}_5]\!]\nu) & \text{otherwise} \end{cases}$$

Since a tile is a tuple of glues, a tile maps to $((\{\texttt{nullGlue}\} \cup \mathbb{Z}) \times \mathbb{Z})^4$, plus `null` for the `noTile` case:

$$\underline{[\![\cdot]\!] :\ \texttt{Tile} \to \texttt{St} \to ((\{\texttt{nullGlue}\} \cup \mathbb{Z}) \times \mathbb{Z})^4 \cup \texttt{null}}$$

$$[\![\texttt{noTile}]\!]\nu = \texttt{null}$$

$$[\![(n, s, e, w)]\!]\nu = ([\![n]\!]\nu, [\![s]\!]\nu, [\![e]\!]\nu, [\![w]\!]\nu)$$

$$[\![\texttt{S}(\texttt{E}_1, \texttt{E}_2)]\!]\nu = \nu(\texttt{S})([\![\texttt{E}_1]\!]\nu, [\![\texttt{E}_2]\!]\nu)$$

TileSets and GlueSets are simply sets of their respective types:

$$\begin{array}{l}
[\![\cdot]\!] : \ \texttt{TileSet} \to \texttt{St} \to 2^{\{\texttt{nullGlue}\}\cup\mathbb{Z})\times\mathbb{Z})^4\cup\texttt{null}} \\
\hline
[\![\emptyset]\!]\nu = \emptyset \\
[\![\{\texttt{T}\}]\!]\nu = \{[\![\texttt{T}]\!]\nu\} \\
[\![\{\texttt{T}\} \cup \texttt{TS}]\!]\nu = \{[\![\texttt{T}]\!]\nu\} \cup [\![\texttt{TS}]\!]\nu
\end{array}$$

$$\begin{array}{l}
[\![\cdot]\!] : \ \texttt{GlueSet} \to \texttt{St} \to 2^{(\{\texttt{nullGlue}\}\cup\mathbb{Z})\times\mathbb{Z}} \\
\hline
[\![\emptyset]\!]\nu = \emptyset \\
[\![\{\texttt{G}\}]\!]\nu = \{[\![\texttt{G}]\!]\nu\} \\
[\![\{\texttt{G}\} \cup \texttt{GS}]\!]\nu = \{[\![\texttt{G}]\!]\nu\} \cup [\![\texttt{GS}]\!]\nu
\end{array}$$

4 Tile Assembly Program

Here we define a program that verifies whether a tile set results in a specific unique assembly. This function is given in terms of smaller helper functions, which are defined in Section 4.2.

4.1 The Verification Program

For a given assembly, called `assembly`, the proof of correctness follows from proving the validity of the following dynamic logic formula, making use of the notation for nondeterministic programing defined in class. The precondition `init` is specific to the tile set. It represents the state in which the seed tile is present at the origin and all other locations have no tiles. The nondeterministic program `assemble` acts as the verifier in that its behavior mimics that of the assembly process. Both of these will be formally defined later.

$$\texttt{init}[\texttt{assemble}]\nu(\texttt{S}) = \texttt{assembly}$$

4.2 Helper Functions and their Semantics

We first establish some smaller functions which we use to define `assemble`. The first two functions, `testLoc` and `glueCount`, determine whether a tile can be placed in a certain location based on the rules of the model. The nondeterministic function `addOne` augments the structure by placing a tile if there is some combination of tile and location that passes the previous tests. Finally, the `assemble` calls `addOne` until no more tiles can be added.

Test Functions First we define a function $\texttt{testLoc} : \texttt{Structure} \times \texttt{Location} \to \{\texttt{true}, \texttt{false}\}$ that tests whether a given location in the structure is empty:

$$\texttt{testLoc} : \texttt{S}, (x, y) \mapsto (\texttt{S}(x, y) = \texttt{noTile}).$$

The semantics of `testLoc` are as follows:

$$\begin{aligned}
[\![\texttt{testLoc}(\texttt{S}(x, y))]\!]\nu &= [\![\texttt{S}(x, y) = \texttt{noTile}]\!]\nu \\
&= \{\nu : \nu(\texttt{S})(x, y) = \texttt{null}\}
\end{aligned}$$

Next, we define a function $\texttt{glueCount} : \texttt{Structure} \times \texttt{Location} \times \texttt{Tile} \to \mathbb{N}$, which calculates the total strength of matching glues given a tile and a position in the structure.

$$\begin{aligned}\texttt{glueCount} :& (\texttt{S}, (x, y), ((n_v, n_s)_g, (s_v, s_s)_g, (e_v, e_s)_g, (w_v, w_s)_g)) \mapsto \\ &\quad \texttt{if } \texttt{S}(x-1, y, 2) = (w_v, w_s)_g \texttt{ and not } w_v = \texttt{nullGlue} \texttt{ then } w_s \texttt{ else } 0 \\ &\quad + \texttt{if } \texttt{S}(x+1, y, 3) = (e_v, e_s)_g \texttt{ and not } e_v = \texttt{nullGlue} \texttt{ then } e_s \texttt{ else } 0 \\ &\quad + \texttt{if } \texttt{S}(x, y+1, 0) = (s_v, s_s)_g \texttt{ and not } s_v = \texttt{nullGlue} \texttt{ then } s_s \texttt{ else } 0 \\ &\quad + \texttt{if } \texttt{S}(x, y-1, 1) = (n_v, n_s)_g \texttt{ and not } n_v = \texttt{nullGlue} \texttt{ then } n_s \texttt{ else } 0\end{aligned}$$

We derive the semantics of $\texttt{glueCount}$ as follows:

$$\begin{aligned}&[\![\texttt{glueCount}(\texttt{S}, (x, y), ((n_v, n_s)_g, (s_v, s_s)_g, (e_v, e_s)_g, (w_v, w_s)_g)))]\!]\nu \\ &= [\![\texttt{if } \texttt{S}(x-1, y, 2) = (w_v, w_s)_g \texttt{ and not } w_v = \texttt{nullGlue} \texttt{ then } w_s \texttt{ else } 0]\!]\nu \\ &+ [\![\texttt{if } \texttt{S}(x+1, y, 3) = (e_v, e_s)_g \texttt{ and not } e_v = \texttt{nullGlue} \texttt{ then } e_s \texttt{ else } 0]\!]\nu \\ &+ [\![\texttt{if } \texttt{S}(x, y+1, 0) = (s_v, s_s)_g \texttt{ and not } s_v = \texttt{nullGlue} \texttt{ then } s_s \texttt{ else } 0]\!]\nu \\ &+ [\![\texttt{if } \texttt{S}(x, y-1, 1) = (n_v, n_s)_g \texttt{ and not } n_v = \texttt{nullGlue} \texttt{ then } n_s \texttt{ else } 0]\!]\nu \\ &= \begin{cases} [\![w_s]\!]\nu & \text{if } \nu(\texttt{S})([\![x-1]\!]\nu, [\![y]\!]\nu, 2) = [\![(w_v, w_s)_g]\!]\nu \text{ and } [\![w_v]\!]\nu \neq \texttt{nullGlue} \\ 0 & \text{otherwise} \end{cases} \\ &+ \begin{cases} [\![e_s]\!]\nu & \text{if } \nu(\texttt{S})([\![x+1]\!]\nu, [\![y]\!]\nu, 3) = [\![(e_v, e_s)_g]\!]\nu \text{ and } [\![e_v]\!]\nu \neq \texttt{nullGlue} \\ 0 & \text{otherwise} \end{cases} \\ &+ \begin{cases} [\![s_s]\!]\nu & \text{if } \nu(\texttt{S})([\![x]\!]\nu, [\![y+1]\!]\nu, 0) = [\![(s_v, s_s)_g]\!]\nu \text{ and } [\![s_v]\!]\nu \neq \texttt{nullGlue} \\ 0 & \text{otherwise} \end{cases} \\ &+ \begin{cases} [\![n_s]\!]\nu & \text{if } \nu(\texttt{S})([\![x]\!]\nu, [\![y-1]\!]\nu, 1) = [\![(n_v, n_s)_g]\!]\nu \text{ and } [\![n_v]\!]\nu \neq \texttt{nullGlue} \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

From these two functions we create a test function $\texttt{test} : \texttt{Structure} \times \texttt{Location} \times \texttt{Tile} \to \{\texttt{true}, \texttt{false}\}$ which determines whether a tile can be added to a given location.

$$\texttt{test} : (\texttt{S}, (x, y), t) \mapsto \texttt{testLoc}(\texttt{S}, (x, y)) \texttt{ and } \texttt{glueCount}(\texttt{S}, (x, y), t) \geq 2$$

Its semantics are as follows:

$$\begin{aligned}[\![\texttt{test}(\texttt{S}, (x, y), t)]\!] &= [\![\texttt{testLoc}(\texttt{S}, (x, y)) \texttt{ and } \texttt{glueCount}(\texttt{S}, (x, y), t) \geq 2]\!] \\ &= [\![\texttt{testLoc}(\texttt{S}, (x, y))]\!] \cap [\![\texttt{glueCount}(\texttt{S}, (x, y), t) \geq 2]\!] \\ &= [\![\texttt{testLoc}(\texttt{S}, (x, y))]\!] \cap \{\nu : [\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!] \geq 2\}\end{aligned}$$

Augmenting Functions The following subroutines nondeterministically add entries to the array representing the structure, which corresponds to the assembly growing with the addition of a new tile. The subroutine $\texttt{addOne}$ nondeterministically adds a tile to an empty position in the structure array if such an addition can be made according to the model.

$$\texttt{addOne} = \bigsqcup_{t \in \texttt{T}} \bigsqcup_{\substack{x \in \mathbb{Z} \\ y \in \mathbb{Z}}} (?\texttt{test}(\texttt{S}, (x, y), t)); \texttt{S}(x, y) := t,$$

where $\sqcup$ denotes the nondeterministic choice operator to avoid confusion with the traditional set union operator. Its semantics follow from those defined on nondeterministic programs.

$$\begin{aligned}
[\![\texttt{addOne}]\!] &= [\![\bigsqcup_{t\in\mathtt{T}} \bigsqcup_{\substack{x\in\mathbb{Z}\\ y\in\mathbb{Z}}} (?\texttt{test}(\mathtt{S},(x,y),t));\mathtt{S}(x,y) := t]\!] \\
&= \bigcup_{\substack{t\in\mathtt{T}\\ x,y\in\mathbb{Z}}} [\![?\texttt{test}(\mathtt{S},(x,y),t))]\!] \circ [\![\mathtt{S}(x,y) := t]\!] \\
&= [\![\texttt{testLoc}(\mathtt{S},(x,y))]\!] \cap \{\nu : [\![\texttt{glueCount}(\mathtt{S},(x,y),t)]\!]\nu \geq 2\} \circ \{(\nu,\omega) : \nu = \omega \text{ except } \omega(\mathtt{S}(x,y) = [\![t]\!]\} \\
&= \{(\nu,\omega) : \nu(\mathtt{S}(x,y) = \texttt{null} \text{ and } [\![\texttt{glueCount}(\mathtt{S},(x,y),t)]\!] \geq 2 \text{ and } \nu = \omega \text{ except } \omega(\mathtt{S})(x,y) = [\![t]\!]\}
\end{aligned}$$

4.3 Semantics of the Verifier

Finally, we define the `assemble` program and its semantics. The `assemble` program continues to add tiles to the structure until no further tiles can be added.

$$\texttt{assemble} = (\texttt{addOne})^*; (?[\texttt{addOne}]\texttt{false}).$$

Its semantics are as follows:

$$\begin{aligned}
[\![\texttt{assemble}]\!] &= [\![(\texttt{addOne})^*; (?[\texttt{addOne}]\texttt{false})]\!] \\
&= [\![\texttt{addOne}^*]\!] \circ [\![?[\texttt{addOne}]\texttt{false}]\!] \\
&= [\![\texttt{addOne}^*]\!] \circ \{(\nu,\nu) : \forall\omega \in \mathtt{St}\ (\nu,\omega) \notin [\![\texttt{addOne}]\!]\} \\
&= [\![\bigsqcup_{\ell\in\mathbb{N}} \texttt{addOne}^{\ell}]\!] \circ \{(\nu,\nu) : \forall\omega \in \mathtt{St}\ (\nu,\omega) \notin [\![\texttt{addOne}]\!]\} \\
&= \bigcup_{\ell\in\mathbb{N}} [\![\texttt{addOne}^{\ell}]\!] \circ \{(\nu,\nu) : \forall\omega \in \mathtt{St}\ (\nu,\omega) \notin [\![\texttt{addOne}]\!]\} \\
&= \bigcup_{\ell\in\mathbb{N}} \bigcup_{x,y\in\mathbb{Z}} \bigcup_{t\in\mathtt{T}} \{(\nu,\omega) : \nu(\mathtt{S})(x,y) = \texttt{noTile} \text{ and } \omega(\mathtt{S})(x,y) = [\![t]\!] \text{ and } \nu = \omega \text{ otherwise} \\
&\qquad \text{and } [\![\texttt{glueCount}(\mathtt{S},(x,y),t)]\!] \geq 2\}^{\ell} \circ \{(\nu,\nu) : \forall\omega \in \mathtt{St}\ (\nu,\omega) \notin [\![\texttt{addOne}]\!]\}
\end{aligned}$$

Finally, we define the precondition `init`, which is satisfied when the specified seed tile, denoted by `seed`, occupies position $(0,0)$ in the structure array and all other positions are empty.

$$\texttt{init} \equiv \mathtt{S}(0,0) = \texttt{seed} \wedge (\forall x,y\ (x = 0 \wedge y = 0) \vee \mathtt{S}(x,y) = \texttt{noTile})$$

The meaning of `init` in our semantics follows directly from our definitions.

$$[\![\texttt{init}]\!] = \{\nu : \nu(\mathtt{S})(0,0) \text{ and } \forall(x,y) \in \mathbb{Z}^2 \setminus \{(0,0)\}, \nu(\mathtt{S})(x,y) = \texttt{noTile}\}$$

5 Proofs

5.1 Terminal Assembly Definition

Our goal in this section is to verify the assembly of an $m\times n$ rectangle. We begin by giving a formal definition of the tile set for this problem and the terminal assembly generated by the tile set.

The terminal assembly is shown in Figure 3. As shown in the figure, there are four types of tiles: the corner, the horizontal top edge tiles, the vertical left edge tiles, and the fill tiles. There are three types of

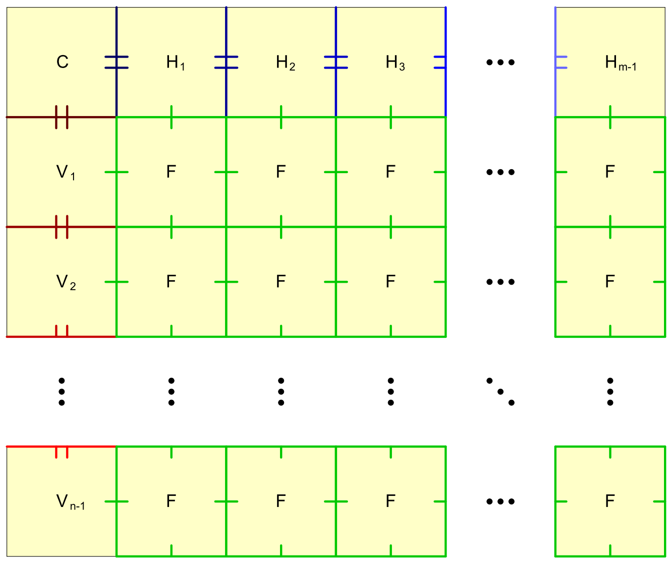


Figure 3: The terminal assembly $\texttt{Rect}_{m,n}$

glues: horizontal glues that bind the horizontal top edge tiles, vertical glues that bind the vertical left edge tiles, and a fill glue.

The corner tile C has null glues on the north and west edges and strength-2 glues on the east and south edges. This tile is the seed for the $m \times n$ rectangle assembly. The horizontal top edge tiles $\texttt{H}_x$ have a null glue on the north edge, a fill glue on the south edge, and horizontal glues on the east and west edges. The vertical left edge tiles $\texttt{V}_y$ have a null glue on the west edge, a fill glue on the east edge, and vertical glues on the north and south edges. The fill tiles F have fill glues on all four edges. These are formally defined as follows:

$$\begin{aligned}
\texttt{C} &= ((\texttt{nullGlue}, 0), (v_1, 2), (h_1, 2), (\texttt{nullGlue}, 0)) \\
\texttt{V}_y &= ((v_y, 2), (v_{y+1}, 2), (g, 1), (\texttt{nullGue}, 0)), && 0 < y < m-1 \\
\texttt{V}_{m-1} &= ((v_{m-1}, 2), (\texttt{nullGlue}, 0), (g, 1), (\texttt{nullGlue}, 0)) \\
\texttt{H}_x &= ((\texttt{nullGlue}, 0), (g, 1), (h_{x+1}, 2), (h_x, 2)), && 0 < x < n-1 \\
\texttt{H}_{n-1} &= ((\texttt{nullGlue}, 0), (g, 1), (\texttt{nullGlue}, 0), (h_x, 2)) \\
\texttt{F} &= ((g, 1), (g, 1), (g, 1), (g, 1)).
\end{aligned}$$

Note that there are actually $m-1$ and $n-1$ vertical and horizontal tiles respectively, each with a different pair of glues. This is to ensure that the $\texttt{V}_{m-1}$ (respectively, $\texttt{H}_{n-1}$) tile is the last tile added to the column (respectively, row), and that it only is added when the column is of height m (respectively, n). The complete tile set is:

$$\texttt{TS} = \{\texttt{C}, \texttt{F}\} \cup \{\texttt{V}_y \ : \ 0 < y < m\} \cup \{\texttt{H}_x \ : \ 0 < x < n\}.$$

The complete glue set is:

$$\texttt{GS} = \{g\} \cup \{v_y \ : \ 0 < y < m-1\} \cup \{h_x \ : \ 0 < x < n-1\}.$$

We are now ready to define the terminal assembly, which is given by an array $\mathtt{Rect}_{m,n}$, defined as follows:

$$\begin{aligned}
&\mathtt{Rect}_{m,n}(0,0) = \mathtt{C} && \\
&\mathtt{Rect}_{m,n}(0,y) = \mathtt{V}_y, && 0 < y < m \\
&\mathtt{Rect}_{m,n}(x,0) = \mathtt{H}_x, && 0 < x < n \\
&\mathtt{Rect}_{m,n}(x,y) = \mathtt{F}, && 0 < y < m \text{ and } 0 < x < n \\
&\mathtt{Rect}_{m,n}(x,y) = \mathtt{noTile}, && \text{otherwise}
\end{aligned}$$

5.2 Proofs of Correctness

In this section, we prove that the `assemble` program correctly assembles an $m \times n$ rectangle when seeded with $\mathtt{S}(0,0) = \mathtt{C}$. This is formalized in Theorem 5.2.

Definition 5.1. We say that the assembly $\mathtt{S}$ is a *substructure* of $\mathtt{Rect}_{m,n}$ if for each $x, y \in \mathbb{Z}$, either $\mathtt{S}(x,y) = \mathtt{noTile}$ or $\mathtt{S}(x,y) = \mathtt{Rect}_{m,n}(x,y)$. Further, $\mathtt{S}$ is called a *strict substructure* of $\mathtt{Rect}_{m,n}$ if it is a substructure of $\mathtt{Rect}_{m,n}$ and there exists some $(x,y) \in \mathbb{Z}^2$ such that $\mathtt{S}(x,y) = \mathtt{noTile}$ and $\mathtt{Rect}_{m,n}(x,y) \neq \mathtt{noTile}$.

Theorem 5.2. $\mathtt{init}[\mathtt{assemble}(\mathtt{S})]\nu(\mathtt{S}) = \mathtt{Rect}_{m,n}$, *where* `init` *is as defined in Section 4.3 and* $\mathtt{Rect}_{m,n}$ *is as defined in Section 5.1.*

Proof. The following three properties hold:

1. At all times throughout running `assemble`, the current assembly is a substructure of the terminal assembly (Theorem 5.3).
2. For any strict substructure of the terminal assembly, a tile can always be added (Lemma 5.4).
3. No tile can be added to the terminal assembly (Lemma 5.5).

The result follows from these three properties. □

The remainder of this section is devoted to proving the three properties given in Theorem 5.2.

Theorem 5.3. *For all* $\ell \in \mathbb{N}$*, for all* $t \in T$*, and for all* $\nu \in [\![\mathtt{init}]\!] \circ [\![\mathtt{addOne}^{\ell}]\!]$*, the following hold:*

1. $\nu(\mathtt{S})$ *is a substructure of* $\mathtt{Rect}_{m,n}$.
2. $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]\nu < 2$ *for all* (x,y) *such that* $x < 0$ *or* $y < 0$ *and for all* (x,y) *such that* $x + y > \ell + 1$
3. $\nu(\mathtt{S})(x,y) = \mathtt{noTile}$ *for all* (x,y) *such that* $x + y > \ell$
4. *For every* x, y *such that* $\nu(\mathtt{S})(x,y) \neq \mathtt{noTile}$ *all* (a,b) *such that* $0 \le a \le x$ *and* $0 \le b \le y$ *have* $\nu(\mathtt{S})(a,b) = \mathtt{Rect}_{m,n}(a,b)$.

Proof. By induction on ℓ. In the base case $\ell = 0$, $\nu \in [\![\mathtt{init}]\!]$. By definition of `init`, $\nu(\mathtt{S})(0,0) = \mathtt{C}$ and $\nu(\mathtt{S})(x,y) = \mathtt{noTile}$ for all (x,y) with $x \neq 0$ or $y \neq 0$, so 1 and 3 follow. To prove 2 and 4, we consider five cases, some of which may overlap.

Case 1: $x < -1$ or $y \neq 0$.
Each of $\nu(\mathtt{S})(x-1,y)$, $\nu(\mathtt{S})(x+1,y)$, $\nu(\mathtt{S})(x,y-1)$, and $\nu(\mathtt{S})(x,y+1)$ is `noTile`, so each summand of $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]\nu = 0$ for all t

Case 2: $x = -1$ and $y = 0$. Each of $\nu(\mathtt{S})(x-1,y)$, $\nu(\mathtt{S})(x,y-1)$, and $\nu(\mathtt{S})(x,y+1)$ is `noTile`. The tile at $\nu(\mathtt{S})(0,y-1) = \mathtt{C}$. Its contribution to the meaning of `glueCount` is zero because its west glue is `nullGlue`. Consequently all summands of $[\![\mathtt{glueCount}(\mathtt{S}(x,y),t)]\!]\nu$ are 0 for every t.

Case 3: $y < -1$ or $x \neq 0$.
Each of $\nu(\texttt{S})(x-1, y)$, $\nu(\texttt{S})(x+1, y)$, $\nu(\texttt{S})(x, y-1)$, and $\nu(\texttt{S})(x, y+1)$ is `noTile`, so each summand of $[\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\nu = 0$ for each t

Case 4: $y = -1$ and $x = 0$ Each of $\nu(\texttt{S})(x-1, y)$, $\nu(\texttt{S})(x, y-1)$, and $\nu(\texttt{S})(x+1, y)$ is `noTile`. The tile at $\nu(\texttt{S})(x, y+1) = \texttt{C}$. Its contribution to the meaning of `glueCount` is zero because its north glue is `nullGlue`. Consequently all summands of $[\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\nu$ are 0 for every t.

Case 5: $x + y > 1$.
Each of $\nu(\texttt{S})(x-1, y)$, $\nu(\texttt{S})(x+1, y)$, $\nu(\texttt{S})(x, y-1)$, and $\nu(\texttt{S})(x, y+1)$ is `noTile`, so each summand of $[\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\nu = 0$ for every t.

Inductively assume the lemma holds for all $\nu \in [\![\texttt{init}]\!] \circ [\![\texttt{addOne}^\ell]\!]$. We show that for all $(\nu, \omega) \in [\![\texttt{addOne}]\!]$, the above conditions are maintained. Recall that for each $(\nu, \omega) \in [\![\texttt{addOne}]\!]$, we have that for some x', y', $\nu(\texttt{S})(x', y') = \texttt{noTile}$, $[\![\texttt{glueCount}(\texttt{S}, (x', y'), t)]\!]\nu \geq 2$, and $\nu = \omega$ except $\omega(\texttt{S})(x', y') = [\![t]\!]\nu$. We consider the following cases for x' and y', the indices of the only changed value in `S`. By the inductive hypothesis, neither x' nor y' can be negative because $[\![\texttt{glueCount}(\texttt{S},(x'y'), t)]\!]\nu < 2$ when x or y is negative. Similarly, $x' + y' \leq \ell + 1$.

Case 1: $x' = 0$.
Based on the induction hypothesis, because $\nu(\texttt{S})(x', y') = \texttt{noTile}$, $\nu(\texttt{S})(x'+1, y') = \nu(\texttt{S})(x'-1, y') = \nu(\texttt{S})(x', y'+1) = \texttt{noTile}$. Additionally $\nu(\texttt{S})(x', y'-1)$ is either `noTile` or $\texttt{V}_{y'-1}$. Because a tile t is being added to the structure, $[\![\texttt{glueCount}(\texttt{S},(x', y'), t)]\!]\nu \geq 2$. The only t for which this is possible is $t = \texttt{V}_{y'}$, as the only matching glue appears on the south side of $\texttt{V}_{y'-1}$. Consequently, $\omega(\texttt{S})(x', y') = \texttt{V}_{y'}$ and `S` remains a substructure of $\texttt{Rect}_{m,n}$.

For all x, y with $y < 0$ or $x < -1$, $[\![\texttt{glueCount}(\texttt{S},(x, y), t)]\!]\nu = [\![\texttt{glueCount}(\texttt{S},(x, y), t)]\!]\omega$, as none of the neighboring entries are changed.

For all x, y with $x = -1$ and $y \geq 0$, it is apparent that $[\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\nu = [\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\omega$ when $(x, y) \neq (-1, y')$. The only entry adjacent to $\omega(\texttt{S})(-1, y')$ with a value other than `noTile` is $\omega(\texttt{S})(x', y')$, but since the tile at this location is $\texttt{V}_{y'}$, the glue on its west edge is `nullGlue`, so it contributes 0 to the value of $[\![\texttt{glueCount}(\texttt{S}, (-1, y'), t)]\!]\omega$.

Finally, the only locations for which $[\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\omega > [\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\nu$ are $(x, y) = (1, y')$ and $(x, y) = (0, y'+1)$. Since $1+y' = 0+y'+1 \leq \ell+2$, the property 2 is preserved. Similarly, the only location with $\omega(\texttt{S})(x, y) \neq \nu(\texttt{S})(x, y)$ is $(x, y) = (0, y')$ and by the induction hypothesis, $y' \leq \ell+1$, so property 3 is also preserved.

Case 2: $y' = 0$.
The case for $y' = 0$ is symmetric to case 1.

Case 3: $x' > 0$ and $y' > 0$. By the induction hypothesis, $\nu(\texttt{S})$ is a substructure of $\texttt{Rect}_{m,n}$. Due to property 4, $\nu(\texttt{S})(x'+1, y') = \nu(\texttt{S})(x', y+1) = \texttt{noTile}$. Because a tile was added to $\texttt{S}(x', y')$ $[\![\texttt{glueCount}(\texttt{S}, (x', y'), t)]\!]\nu \geq 2$. The only tile for which this is possible is $t = \texttt{F}$, as it is the only tile with glue g on its north and west sides. Consequently $\omega(\texttt{S})$ remains a substructure of $\texttt{Rect}_{m,n}$.

Because location (x', y') is not adjacent to any location with $x < 0$ or $y < 0$, $[\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\omega = [\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\nu$ for all (x, y) with $x < 0$ or $y < 0$. The only (x, y) such that $[\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\omega > [\![\texttt{glueCount}(\texttt{S}, (x, y), t)]\!]\nu$ are $(x, y) = (x'+1, y')$ and $(x, y) = (x', y'+1)$. Since $x' + y' \leq \ell + 1$, both $x' + 1 + y' \leq \ell + 2$ and $x + y' + 1 \leq \ell + 2$, so property 2 is preserved. Property 3 also follows from the fact that $x' + y' \leq \ell + 1$ and (x', y') is the only (x, y) for which $\omega(\texttt{S})(x, y) \neq \nu(\texttt{S})(x, y)$.

Finally, because `F` attached at (x', y') via glues on its north and west sides, both $\nu(\texttt{S})(x'-1, y)$ and $\nu(\texttt{S})(x, y-1)$ were not `noTile`. Applying the induction hypothesis, property 4 is maintained.

□

Lemma 5.4. *If the assembly is not terminal, then a tile can be added.*

Proof. From Theorem 5.3, we know that if the assembly is not terminal, then it is a substructure with the property described in Theorem 5.3 part 4. Specifically, it has the property that if a tile has been added to location (x, y), then there are tiles at all locations (x', y') with $0 \le x' \le x$ and $0 \le y' \le y$. There are two possible forms for such a substructure, and we consider them separately.

Case 1: The substructure is a rectangle; that is, there is only one location (x, y) such that $\texttt{S}(x+1, y) = \texttt{noTile}$ and $\texttt{S}(x, y+1) = \texttt{noTile}$. Then either $x < n-1$ or $y < m-1$. Suppose $x < n-1$. Then

$$\begin{aligned}[\![\texttt{glueCount}(\texttt{S}, (x+1, y), \texttt{H}_{x+1})]\!] &= [\![\texttt{glueCount}(\texttt{S}, (x+1, y), ((\texttt{nullGlue}, 0), (g, 1), (h_{x+2}, 2), (h_{x+1}, 2))]\!] \\ &= 2 + 0 + 0 + 0 \\ &= 2,\end{aligned}$$

so

$$\begin{aligned}[\![\texttt{test}(\texttt{S}, (x+1, y), \texttt{H}_{x+1})]\!] &= [\![\texttt{testLoc}(\texttt{S}, (x+1, y))]\!] \cap \{\nu : [\![\texttt{glueCount}(\texttt{S}, (x+1, y), \texttt{H}_{x+1})]\!] \ge 2\} \\ &= [\![\texttt{true}]\!] \cap [\![\texttt{true}]\!].\end{aligned}$$

Suppose instead that $y < m-1$. Then

$$\begin{aligned}[\![\texttt{glueCount}(\texttt{S}, (x, y+1), \texttt{V}_{y+1})]\!] &= [\![\texttt{glueCount}(\texttt{S}, (x, y+1), ((v_{y+1}, 2), (v_{y+2}, 2), (g, 1), (\texttt{nullGlue}, 0))]\!] \\ &= 0 + 0 + 0 + 2 \\ &= 2,\end{aligned}$$

so

$$\begin{aligned}[\![\texttt{test}(\texttt{S}, (x, y+1), \texttt{V}_{y+1})]\!] &= [\![\texttt{testLoc}(\texttt{S}, (x, y+1))]\!] \cap \{\nu : [\![\texttt{glueCount}(\texttt{S}, (x, y+1), \texttt{V}_{y+1})]\!] \ge 2\} \\ &= [\![\texttt{true}]\!] \cap [\![\texttt{true}]\!] \\ &= \texttt{St}.\end{aligned}$$

Hence if the substructure is a rectangle, it is possible to add a tile.

Case 2: The substructure is not a rectangle; that is, there are at least two locations (x_1, y_1) and (x_2, y_2) with $x_1 > x_2$ and $y_1 < y_2$ such that $\texttt{S}(x_1+1, y_1) = \texttt{noTile}$ and $\texttt{S}(x_1, y_1+1) = \texttt{noTile}$ and $\texttt{S}(x_2+1, y_2) = \texttt{noTile}$ and $\texttt{S}(x_2, y_2+1) = \texttt{noTile}$. Then there is some location (x_2+1, y'), $y_1 < y' < y_2$ where a tile of type $\texttt{F}$ can be added. We know there is no tile at location (x_2+1, y_2) by definition. If there is a tile at location (x_2+1, y_2-1), then the glue on its south edge must be $(g, 1)$ because the structure is a substructure of the terminal assembly, and all internal glues in the terminal assembly are $(g, 1)$. So if $\texttt{S}(x_2+1, y_2-1) \ne \texttt{noTile}$, then

$$\begin{aligned}[\![\texttt{glueCount}(\texttt{S}, (x_2+1, y_2), \texttt{F})]\!] &= [\![\texttt{glueCount}(\texttt{S}, (x_2+1, y_2), ((g, 1), (g, 1), (g, 1), (g, 1))]\!] \\ &= 1 + 0 + 0 + 1 \\ &= 2,\end{aligned}$$

so $[\![\texttt{test}(\texttt{S}, (x_2+1, y_2), \texttt{F})]\!] = \texttt{St}$. Hence we can add a tile at location (x_2+1, y_2). If there is no tile at location (x_2+1, y_2-1) but there is a tile at location (x_2+1, y_2-2), then we follow the same reasoning to determine that we can add a tile at location (x_2+1, y_2-1). Otherwise, we continue vertically until we find a tile at the location immediately above the location at which we are trying to add a tile. We know that we will eventually find such a tile because due to Theorem 5.3, we know there is a tile at location (x_2+1, y_1).

For both types of substructures, we can add a tile. Hence if the current assembly is a substructure of the terminal assembly, it is possible to add a tile. □

Lemma 5.5. $\mathtt{S} = \mathtt{Rect}_{m,n}[\mathtt{test}((x,y),t)]\mathtt{false}$. *That is, no tile can be added to the terminal assembly.*

Proof. We consider eight cases for the location (x,y).

Case 1: $x < -1$ or $y < -1$ or $x > n$ or $y > m$. Then $\mathtt{S}(x-1,y)$, $\mathtt{S}(x+1,y)$, $\mathtt{S}(x,y-1)$, and $\mathtt{S}(x,y+1)$ are all `noTile`, so $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!] = 0$ for all tiles t. Hence a tile cannot be added at any of these locations.

Case 2: $x = -1$ and $0 \le y < m$. Then $\mathtt{S}(x-1,y)$, $\mathtt{S}(x,y-1)$, and $\mathtt{S}(x,y+1)$ are all `noTile` and so contribute 0 to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$. $\mathtt{S}(x+1,y) = \mathtt{V}_y$, but $\mathtt{S}(x+1,y,3) = \mathtt{nullGlue}$ so the contribution to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is still 0 for all tiles t. Hence the total value of $[\![\mathtt{glueCount}(\mathtt{S}(x,y),t)]\!]$ is 0, so a tile cannot be added.

Case 3: $y = -1$ and $0 \le x < n$. Then $\mathtt{S}(x-1,y)$, $\mathtt{S}(x+1,y)$, and $\mathtt{S}(x,y-1)$ are all `noTile` and so contribute 0 to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$. $\mathtt{S}(x,y+1) = \mathtt{H}_x$, but $\mathtt{S}(x,y+1,0) = \mathtt{nullGlue}$ so the contribution to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is still 0 for all tiles t. Hence the total value of $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is 0, so a tile cannot be added.

Case 4: $x = 0$ and $y = m$. Then $\mathtt{S}(x+1,y)$, $\mathtt{S}(x-1,y)$, and $\mathtt{S}(x,y+1)$ are all `noTile` and so contribute 0 to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ for all tiles t. $\mathtt{S}(x,y-1) = \mathtt{V}_{m-1}$, but $\mathtt{S}(x,y-1,1) = \mathtt{nullGlue}$, so the contribution to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is still 0 for all tiles t. Hence the total value of $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is 0, so a tile cannot be added.

Case 5: $x = n$ and $y = 0$. Then $\mathtt{S}(x+1,y)$, $\mathtt{S}(x,y-1)$, and $\mathtt{S}(x,y+1)$ are all `noTile` and so contribute 0 to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ for all tiles t. $\mathtt{S}(x-1,y) = \mathtt{H}_{n-1}$, but $\mathtt{S}(x-1,y,2) = \mathtt{nullGlue}$, so the contribution to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is still 0 for all tiles t. Hence the total value of $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is 0, so a tile cannot be added.

Case 6: $0 < x < n$ and $y = m$. Then $\mathtt{S}(x-1,y)$, $\mathtt{S}(x+1,y)$, and $\mathtt{S}(x,y+1)$ are all `noTile` and so contribute 0 to $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ for all tiles t. $\mathtt{S}(x,y-1) = \mathtt{F}$ and so $\mathtt{S}((x,y-1),1) = (g,1)$. This contributes at most 1 to the value of $[\![\mathtt{glueCount}(\mathtt{S}(x,y),t)]\!]$ if tile t has glue $(g,1)$ on its top edge. Hence the value of $[\![\mathtt{glueCount}(\mathtt{S}(x,y),t)]\!]$ is at most 1, which is not enough to bind a tile. So no tile can be added at location (x,y).

Case 7: $x = n$ and $0 < y < m$. Then $\mathtt{S}(x+1,y)$, $\mathtt{S}(x,y-1)$, and $\mathtt{S}(x,y+1)$ are all `noTile` and so contribute 0 to $[\![\mathtt{glueCount}(\mathtt{S}(x,y),t)]\!]$ for all tiles t. $\mathtt{S}(x-1,y) = \mathtt{F}$ and so $\mathtt{S}((x,y-1),2) = (g,1)$. This contributes at most 1 to the value of $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ if tile t has glue $(g,1)$ on its left edge. Hence the value of $[\![\mathtt{glueCount}(\mathtt{S},(x,y),t)]\!]$ is at most 1, which is not enough to bind a tile. So no tile can be added at location (x,y).

Case 8: $0 \le x < n$ and $0 \le y < m$. Then in the terminal assembly, $\mathtt{S}(x,y) \neq \mathtt{noTile}$, so $\mathtt{testLoc}(\mathtt{S},(x,y))$ fails, and a tile cannot be added.

For all locations (x,y), a tile cannot be added at (x,y). Hence no tile can be added to the terminal assembly. □

6 Conclusion

Our work in this project represents a significant step towards verifying the correctness of tile self-assemblies. We developed a programming language that enables us to represent the components of the tile assembly problem. We defined a semantics for the new components in our language, then wrote a program to verify the correctness of a tile set. We determined the semantics of this program and used it to verify the correctness of a tile set that forms an $m \times n$ rectangle.

This is the first step towards verification of a more general class of problems. There are a number of possible extensions to this work. First, we could use our existing program to verify additional tile sets other than the $m \times n$ rectangle. More generally, we consider only a particular tile assembly model in which only a single tile can be added to the assembly at a time. A more general tile assembly model, the 2-Handed Assembly Model (2HAM) allows two assemblies of multiple tiles to join together. An interesting direction would be to extend our programming language and results to the 2HAM model. Finally, the tile assembly model itself is a somewhat restricted model of biological processes. For example, the tile assembly model assumes that tiles cannot rotate, that all tiles are the same size and shape, and that assembly occurs in a square lattice on the plane. One could imagine a more general model for DNA computing that lifts some of these restrictions. Developing a language and semantics for such a model would present many new interesting challenges.

References

[Adl94] LM Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[Win98] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.