EPiC Series in Computing
Volume 39, 2016, Pages 107–116
SCSS 2016. 7th International Symposium on Symbolic Computation in Software Science

Verification of a brick Wang tiling algorithm

Toshiaki Matsushima[1]*, Yoshihiro Mizoguchi[2], and Alexandre Derouet-Jourdan[3]

[1] Graduate School of Mathematics, Kyushu University
[2] Institute of Mathematics for Industry, Kyushu University
[3] OLM Digital Inc.

Abstract

We have implemented a certified Wang tiling program for tiling a rectangular region using a brick Wang tile set. A brick Wang tile set is a special Wang tile set introduced in 2015 by A. Derouet-Jourdan et al. in computer graphics to model the texture of wall patterns. We have implemented a tiling algorithm using the Coq proof assistant and have presented its proof, which assures the existence of a tiling of any brick Wang tile set for a rectangle of any size. The essential points of the proof are the existence of a tiling for a 2×2 rectangle and a simple induction process. Since the brick Wang tile set is a set of tiles of infinite types, the proof is not straightforward and there are many conditional branches in the proof of the algorithm. The certification with Coq assures that there are no lack of conditions.

1 Introduction

Wang tiles are a class of formal systems introduced by Hao Wang in 1961 [10]. A model of a Wang tile is a square tile with a color on each side (cf. Figure 1). We arrange Wang tiles side by side, while matching the edge colors. Given a finite set of Wang tiles, we consider tilings of the (infinite) Euclidean plane using arbitrarily many copies of the tiles without rotations in the given tile set. An example of a tiling of the plane is shown in Figure 2.

Whether a given finite set of Wang tiles can tile the plane is a decidability problem called the domino problem, and this problem was proved to be undecidable by R. Berger in 1966 [2]. However, the problem is decidable for a bounded region, and if a tile set can tile a given rectangular region for any boundary colors, it can tile the entire plane.

Wang [10] also conjectured that, if a finite set of Wang tiles can tile the plane, then the tiling is periodic. The conjecture was disproved in 1966 by Berger [2], who gave a concrete tile set of 20,426 tiles that always tiles the plane aperiodically. In 1996, J. Kari presented an example of a tile set with only 14 tiles from the viewpoint of cellular automata [7]. Following J. Kari, K. Culik introduced a tile set with 13 tiles in 1996 [4]. Finally, E. Jeandel proved that a tile set with 11 tiles satisfies this condition in 2015 [6], and this is the smallest tile set that can make an aperiodic tiling.

*t.matsushima@kyudai.jp

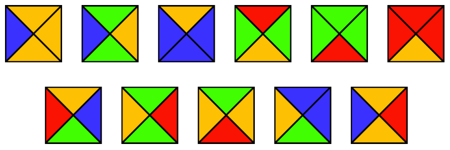


Figure 1: Wang Tiles.

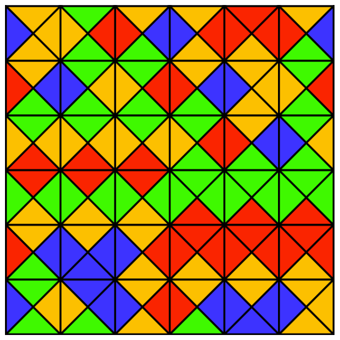


Figure 2: Tiling.

Since a Wang tile set is a simple system that produces aperiodic patterns, there are several applications in computer graphics [3, 8, 9]. In 2015, A. Derouet-Jourdan et al. introduced a class of Wang tiles in computer graphics to model wall pattern textures [5], which is referred to as a brick Wang tile set. Each tile represents a possible way to connect four bricks, where a brick is an element of the wall. It is assumed that the edges of the bricks are axis aligned and that each tile is traversed by a straight line, either vertically or horizontally. In other words, two bricks are always aligned on each tile. For aesthetic reasons, crosses are forbidden, where all four bricks are aligned and the corresponding tile is traversed by two straight lines. In this model, the color in the Wang tiling model is the position of the edge of the brick on the edge of the tile. An example of such a tile is given in Figure 3. In Figure 4, we show a tiling (left) and the corresponding texture as a wall (right), where the colors of the bricks are chosen randomly. In the tiling, the tiles are delimited by the dashed lines, and the bricks are delimited by the red lines. In the texture, the tiles are no longer represented, and the bricks are colored randomly and independently of their size and position.

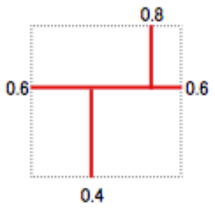


Figure 3: Example of a brick Wang tile represented with the brick edges (red). The colors are represented by numbers and correspond to the positions of the brick edges on the edges of the tile.

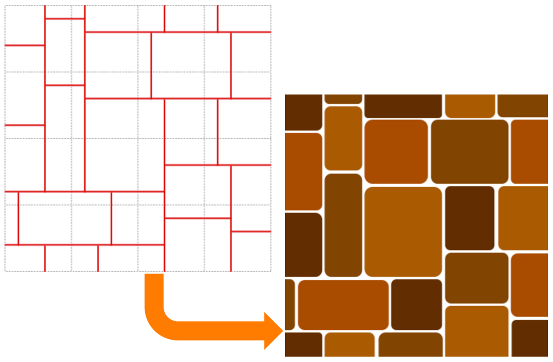


Figure 4: Example of a tiling (left). The tiles and the bricks are delimited by respectively the dashed lines and the red lines. On the right, we transformed the tiling into a brick structure by removing the tiles edges, adding random colors to the bricks, and rounding the corners of the bricks.

If we use a brick Wang tile set for tiling, we can tile any rectangular region larger than 2 squares by 2 squares with any boundary color conditions. An inductive proof has been presented in a previous paper [5].

Since the brick Wang tile set is a set of tiles of infinite types, the proof is not straightforward and there are many conditional branches in the proof. Thus, we have tried to provide a machine-verifiable proof using the Coq proof assistant system [1]. We formalized the notion of Wang tiles in Coq and implemented an algorithm to construct a valid tiling for a given rectangular region and boundary condition as a function in Coq. Furthermore, we proved the correctness of the function using Coq. The correctness and totality of the tiling function provides a constructive proof of the existence of a tiling.

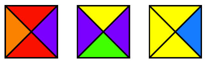
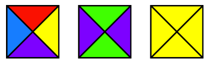
2 Brick Wang tiling

Let $C = \{i \,|\, 0 \le i \le k\}$ $(k > 0)$ be a finite set of colors.

Definition 1 (Wang Tile)**.** *A* **Wang tile** *is a function* $w : \{t, l, b, r\} \to C$.

Definition 2 (Brick Wang Tile)**.** *A Wang tile* w *is said to be a* **brick Wang tile** *if* $w(t) \ne w(b) \wedge w(l) = w(r)$ *or* $w(t) = w(b) \wedge w(l) \ne w(r)$. *We define the set* W_C *of all brick Wang tiles for a given color set* C.

Values $w(t)$, $w(l)$, $w(b)$, and $w(r)$ denote the colors of the top, left, bottom, and right edges, respectively, of a tile w. The following are examples of correct and incorrect brick Wang tiles:

Correct:  Incorrect: 

where $C = \{$🟥, 🟧, 🟨, 🟩, 🟦, 🟪$\}$.

Definition 3. *A* **rectangular region** $\{(i, j) \in \mathbb{N} \times \mathbb{N} \,|\, 1 \le i \le n,\ 1 \le j \le m\}$ *is denoted by* P_{nm}. *A* **tiling** *is a function* $\mathcal{T} : P_{nm} \to W_C$, *and we denote* $\mathcal{T}(i, j)$ *by* $\mathcal{T}_{i,j}$.

Definition 4. *Let* P_{nm} *be a rectangular region. A* **boundary (coloring)** *over* P_{nm} *is a function* $b_{nm} : \{0, n+1\} \times \{j \,|\, 1 \le j \le m\} \cup \{i \,|\, 1 \le i \le n\} \times \{0, m+1\} \to C$.

Definition 5. *Let* P_{nm} *be a rectangular region, and let* b_{nm} *be a boundary coloring over* P_{nm}. *A tiling* $\mathcal{T}$ *is* **valid** *for* b_{nm} *if*

$$\begin{aligned}
\mathcal{T}_{i,j}(t) &= \begin{cases} b_{nm}(0, j) & (i = 1, 1 \le j \le m) \\ \mathcal{T}_{i-1,j}(b) & (2 \le i \le n, 1 \le j \le m), \end{cases} \\
\mathcal{T}_{n,j}(b) &= b_{nm}(n+1, j) \quad (1 \le j \le m), \\
\mathcal{T}_{i,j}(l) &= \begin{cases} b_{nm}(i, 0) & (1 \le i \le n, j = 1) \\ \mathcal{T}_{i,j-1}(r) & (1 \le i \le n, 2 \le j \le m),\ \textit{and} \end{cases} \\
\mathcal{T}_{i,m}(r) &= b_{nm}(i, m+1) \quad (1 \le i \le n).
\end{aligned}$$

Definition 6. *For a valid tiling* $\mathcal{T}$ *for* b_{nm} *over* P_{nm}, *we define* **horizontal edge function** $e_{\mathcal{T}} : \mathbb{N} \to \mathbb{N} \to C$ *and* **vertical edge function** $e'_{\mathcal{T}} : \mathbb{N} \to \mathbb{N} \to C$ *as follows:*

$$e_{\mathcal{T}}(i, j) = \begin{cases} \mathcal{T}_{i+1,j}(t) & (0 \le i \le n-1,\ 1 \le j \le m) \\ \mathcal{T}_{n,j}(b) & (i = n,\ 1 \le j \le m) \\ 0 & (\textit{otherwise}) \end{cases}$$

$$e'_{\mathcal{T}}(i,j) \quad = \quad \begin{cases} \mathcal{T}_{i,j+1}(l) & (1 \le i \le n,\ 0 \le j \le m-1) \\ \mathcal{T}_{i,m}(r) & (1 \le i \le n,\ j = m) \\ 0 & (otherwise) \end{cases}$$

Lemma 1. *Let $\mathcal{T}$ be a valid tiling for a boundary coloring b_{nm} over a rectangular region P_{nm}.*

1. $e_{\mathcal{T}}(0,j) = b(0,j) \quad (1 \le j \le m)$.
2. $e_{\mathcal{T}}(n,j) = b(n+1,j) \quad (1 \le j \le m)$.
3. $e'_{\mathcal{T}}(i,0) = b(i,0) \quad (1 \le i \le n)$.
4. $e'_{\mathcal{T}}(i,m) = b(i,m+1) \quad (1 \le i \le n)$.

Note that there is a technical adjustment of indices. For example, in Figure 6, if we consider $e(i,j) = b_{23}(i,j)$ and $e'(i,j) = b_{23}(i,j)$, then $b_{23}(2,3)$ can be assigned for both $e(2,3)$ and $e'(2,3)$. Therefore, the domain of a boundary coloring b_{nm} is not $\{1,n\} \times \{j \,|\, 1 \le j \le m\} \cup \{i \,|\, 1 \le i \le n\} \times \{1,m\}$ but $\{0,n+1\} \times \{j \,|\, 1 \le j \le m\} \cup \{i \,|\, 1 \le i \le n\} \times \{0,m+1\}$.

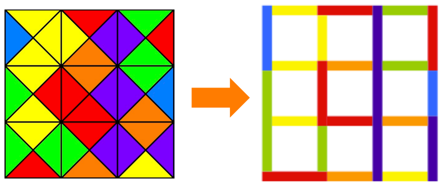


Figure 5: We can consider Wang tiling as edge coloring.

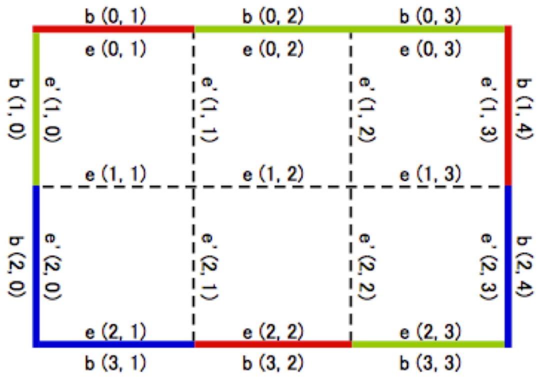


Figure 6: Edge functions and boundaries.

Definition 7. *Let P_{nm} be a rectangular region, $e : \mathbb{N} \to \mathbb{N} \to C$ and $e' : \mathbb{N} \to \mathbb{N} \to C$. A pair (e,e') is a* **brick pair** *over P_{nm}, if $(e(i-1,j) = e(i,j) \wedge e'(i,j-1) \ne e'(i,j))$, or $(e(i-1,j) \ne e(i,j) \wedge e'(i,j-1) = e'(i,j))$ for $1 \le i \le n$, $1 \le j \le m$.*

Proposition 8. *Let P_{nm} be a rectangular region, and let (e,e') be a brick pair over P_{nm}. Then, there exists a boundary coloring b_{nm} over P_{nm} and a valid tiling $\mathcal{T}$ over P_{nm} for b_{nm} such that $(e_{\mathcal{T}}, e'_{\mathcal{T}}) = (e,e')$.* □

We denote a boundary coloring b_{nm} and a tiling $\mathcal{T}$ in Proposition 8 by $b_{nm}(e,e')$ and $\mathcal{T}(e,e')$, respectively.

Definition 9 (Tileable)**.** *Let n and m be natural numbers. Then, P_{nm} is* **tileable** *if there exists a valid tiling for any boundary coloring b_{nm} over P_{nm}.*

Proposition 10 ([5])**.**

1. *P_{12} is not tileable (i.e., there exists a boundary coloring that cannot be satisfied).*
2. *P_{22} is not tileable if $|C| = 2$.*

□

The property in the next Lemma 2 is referred to as the **three-boundary condition**. That is, for any given three colors and edges, there exists a tile in W_C which satisfies those three conditions.

Lemma 2. *Let W_C be the set of all brick Wang tiles for a given color set C. Then we have $\{(w(l), w(t), w(r)) \mid w \in W_C\} = \{(w(t), w(r), w(b)) \mid w \in W_C\} = \{(w(r), w(b), w(l)) \mid w \in W_C\} = \{(w(b), w(l), w(t)) \mid w \in W_C\} = C^3$.* □

Lemma 3. *Let W_C be the set of all brick Wang tiles for a given color set C. Let $n \geq 2$, and let $m \geq 2$. Let P_{nm} be a rectangular region, and let b_{nm} be a boundary coloring over P_{nm}.*

1. *There exist $w_j \in W_C$ $(1 \leq j \leq m)$, such that $w_j(t) = b_{nm}(0, j)$, $w_1(l) = b_{nm}(1, 0)$, and $w_m(r) = b_{nm}(1, m+1)$.*
2. *There exist $w_i \in W_C$ $(1 \leq i \leq n)$, such that $w_i(l) = b_{nm}(i, 0)$, $w_1(t) = b_{nm}(0, 1)$, and $w_n(b) = b_{nm}(n+1, 1)$.*
3. *P_{22} is tileable.*

□

Theorem 1 ([5])**.** *If $|C| \geq 3$, then a rectangular region P_{nm} is tileable for any $n \geq 2$ and $m \geq 2$.* □

Example 1. *Let the color set $C = \{0.2, 0.4, 0.6, 0.8\}$. Consider a brick Wang tile in Figure 3, where $w(l) = w(r) = 0.6$, $w(t) = 0.8$, and $w(b) = 0.4$. We draw three lines on a square tile, the vertices of which are $\{(0,0), (0,1), (1,1), (1,0)\}$ according to the values of $w(l)$, $w(r)$, $w(t)$, and $w(b)$. Since $w(l) = w(r)$, we draw a line from left to right. For this case, we draw a line from $(0, 0.6)$ to $(1, 0.6)$. Since $w(t) \neq w(b)$, we draw two perpendicular lines from $(0.8, 1)$ and $(0.4, 0)$ to the horizontal line. By tiling these brick Wang tiles, a brick wall pattern texture having different size of bricks can be created (cf. Figure 4).*

3 Tiling algorithm

In this section, we introduce a tiling algorithm of brick Wang tiles for a given boundary coloring. We define those functions and proofs in Coq[1].

To simplify the arguments in the following, we consider a color set C as a set of natural numbers, such as $\{i \mid 0 \leq i \leq k\}$, where k is the number of elements in C. In this section, we assume $|C| \geq 3$. We use a notation $\neg c$ to indicate a color different from $c \in C$. That is, we define $\neg c = \min\{c' \mid c' \neq c\}$ and $\neg c_1 \wedge \neg c_2 = \min\{c' \mid c' \neq c_1 \wedge c' \neq c_2\}$. Furthermore, we consider the domain of an every function as `nat` a type of natural numbers $\mathbb{N} = \{0, 1, 2, ...\}$. It is easy to use and check the extracted functions in several programming languages. In other words, we consider that a boundary coloring b, a horizontal edge function e, and a vertical edge function e' all have a type `nat -> nat -> nat`. To describe properties over a finite set C and a finite region P_{nm}, we use some bounding conditions for a variable in `nat`. We do not care about a values of functions b, e and e' outside of the region P_{nm}.

Our algorithm solve a problem over a region P_{nm} for a given boundary condition b_{nm}. That is it finds a brick pair (e, e') from a given boundary coloring b_{nm} over a region P_{nm} such that $\mathcal{T}(e, e')$ is valid. Our algorithm consists of three parts to reduce a tiling problem over P_{nm} to P_{2m} **(Step 1)**, to reduce a tiling problem over P_{2m} to P_{22} **(Step 2)**, and to solve a tiling problem over P_{22} **(Step 3)** (cf. Figure 7).

(Step 1) Let $n, m > 2$. Consider a tiling problem over P_{nm} for a given boundary b_{nm}. We divide it into two tiling problems over P_{1m} for b_{1m} and $P_{(n-1)m}$ for $b_{(n-1)m}$, where

$$b_{1m}(i,j) = \begin{cases} b_{nm}(i,j) & (i = 0, 1), \\ b_{nm}(0,j) & (i = 2, j = 1, 2), \\ \neg\, b_{nm}(0,j) & (i = 2, 3 \leq j \leq m), \end{cases}$$

$$b_{(n-1)m}(i,j) = \begin{cases} b_{1m}(2,j) & (i = 0), \\ b_{nm}(i+1,j) & (i > 0). \end{cases}$$

For a tiling over P_{1m}, we solve it by

$$e(i,j) = \begin{cases} b_{1m}(0,j) & (i = 0), \\ b_{1m}(2,j) & (i = 1), \end{cases}$$

$$e'(1,j) = \begin{cases} b_{nm}(1,0) & (j = 0), \\ \neg\, b_{nm}(1,0) \wedge \neg\, b_{nm}(1,m+1) & (j = 1), \\ b_{nm}(1,m+1) & (2 \leq j \leq m+1). \end{cases}$$

By inductions, we have a tiling problem over P_{2m}.

(Step 2) Let $m > 2$. Consider a tiling over P_{2m} for a given boundary b_{2m}. We divide it into two tiling problem over P_{21} for b_{21} and $P_{2(m-1)}$ for $b_{2(m-1)}$, where

$$b_{21}(i,j) = \begin{cases} b_{2m}(i,j) & (j = 0, 1), \\ b_{2m}(i,0) & (j = 2), \end{cases}$$

$$b_{2(m-1)}(i,j) = \begin{cases} b_{21}(i,2) & (j = 0), \\ b_{2m}(i,j+1) & (j > 0), \end{cases}$$

For a tiling over P_{21}, we solve it by

$$e(i,1) = \begin{cases} b_{2m}(0,1) & (i = 0), \\ \neg\, b_{2m}(0,1) \wedge \neg\, b_{2m}(3,1) & (i = 1), \\ b_{2m}(3,1) & (i = 2), \end{cases}$$

$$e'(i,j) = \begin{cases} b_{21}(i,0) & (j = 0), \\ b_{21}(i,2) & (j = 1). \end{cases}$$

By inductions, we have a tiling problem over P_{22}.

(Step 3) Consider a tiling over P_{22} for a given boundary b. We abbreviate $b_{ij} = b(i,j)$, $e_{ij} = e(i,j)$ and $e'_{ij} = e(i,j)$. Since it satisfies boundary conditions, we have $e_{0j} = b_{0j}$, $e_{2j} = b_{3j}$, $e'_{i0} = b_{i0}$ and $e'_{i2} = b_{i3}$ $(i = 0, 1, j = 0, 1)$. So all we need is defining colors e_{11}, e_{12}, e'_{11} and e'_{21}. According to the equality conditions of $b_{10} = b_{13}$, $b_{20} = b_{23}$, $b_{01} = b_{31}$ and $b_{02} = b_{32}$, we define those values for e and e'. We summarize them in Table 1.

A Coq implementation of (e, e') is a pair of functions `e_nm` and `e'_nm`, as shown in Figure 8. `e_nm` returns a horizontal edge function e for b_{nm}, and `e'_nm` returns a vertical edge function e'. A function `tiling_nm` solves a tiling problem using `e_nm` and `e'_nm`, and makes an array from the pair of edge functions. After extracting functions `e_nm` and `e'_nm` to an OCaml source, we can draw and check solutions for examples of tiling problems using extra rendering functions in OCaml (cf. Appendix).

$b_{10} = b_{13}$	$b_{20} = b_{23}$	$b_{01} = b_{31}$	$b_{02} = b_{32}$	e_{11}	e_{12}	e'_{11}	e'_{21}
yes	yes	-	-	$\neg b_{01} \wedge \neg b_{31}$	$\neg b_{02} \wedge \neg b_{32}$	b_{10}	b_{20}
yes	no	yes	yes	b_{01}	b_{02}	$\neg b_{10}$	$\neg b_{20} \wedge \neg b_{23}$
yes	no	yes	no	b_{01}	b_{02}	$\neg b_{10}$	b_{23}
yes	no	no	yes	b_{31}	$\neg b_{02}$	b_{10}	b_{23}
yes	no	no	no	b_{31}	b_{32}	b_{10}	$\neg b_{20} \wedge \neg b_{23}$
no	yes	yes	yes	b_{31}	b_{32}	$\neg b_{10} \wedge \neg b_{13}$	$\neg b_{20}$
no	yes	yes	no	b_{31}	b_{32}	b_{13}	$\neg b_{20}$
no	yes	no	yes	b_{01}	$\neg b_{02} \wedge \neg b_{32}$	b_{13}	b_{20}
no	yes	no	no	b_{01}	b_{02}	$\neg b_{10} \wedge \neg b_{13}$	b_{20}
no	no	yes	yes	b_{01}	b_{32}	$\neg b_{10} \wedge \neg b_{13}$	$\neg b_{20} \wedge \neg b_{23}$
no	no	yes	no	b_{01}	b_{32}	b_{13}	$\neg b_{20} \wedge \neg b_{23}$
no	no	no	yes	b_{01}	b_{32}	$\neg b_{10} \wedge \neg b_{13}$	b_{20}
no	no	no	no	b_{01}	b_{32}	b_{13}	b_{20}

Table 1: A tiling over P_{22} for a boundary b.

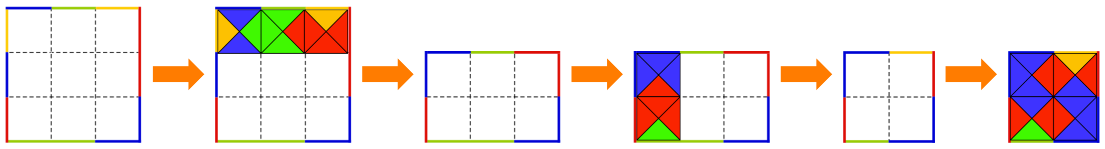


Figure 7: Outline of our algorithm.

4 Verification using Coq

Using Coq, we can verify the properties of the functions we implemented. The property of a valid tiling is divided into three conditions, as follows: a condition for vertical boundary edges (`Boundary_i`), a condition for horizontal boundary edges (`Boundary_j`), and a condition for a brick Wang tile set (`Brick`). The formalized definitions in Coq are shown in Figure 9.

```
Fixpoint e_nm (n m : nat) : boundary -> edge :=
 fun b : boundary =>
 match n with
   | 0 | 1 => e_1m b
   | 2 => enm_to_emn (fun b' => e'_n2 m b') b
   | S n' => fun (i j : nat) =>
               match i with
                 | 0 => (bSnm_to_b1m m b) 0 j
                 | S i' => e_nm n' m (bSnm_to_bnm m b) i' j
               end
 end.

Fixpoint e'_nm (n m : nat) : boundary -> edge
    .... Similar to e_nm ....
end.

Definition tiling_nm (n m : nat)(b : boundary) :=
 tiling n m b (e_nm n m) (e'_nm n m).
```

Figure 8: Main functions e, e', and *tiling*

```
Definition Boundary_i (n m : nat)(b : boundary)(e' : edge) :=
 forall i : nat, e' i 0 == b i 0 /\ e' i m == b i (S m) \/ i = 0 \/ n < i.
Definition Boundary_j (n m : nat)(b : boundary)(e : edge) :=
 forall j : nat, e 0 j == b 0 j /\ e n j == b (S n) j \/ j = 0 \/ m < j.
Definition Brick (n m : nat)(e e' : edge) :=
 forall i j : nat,
 (e i (S j) == e (S i) (S j) /\ e' (S i) j != e' (S i) (S j)) \/
 (e i (S j) != e (S i) (S j) /\ e' (S i) j == e' (S i) (S j)) \/
 n <= i \/ m <= j.
Definition Valid (n m : nat)(b : boundary)(e e' : edge) :=
 Boundary_i n m b e' /\ Boundary_j n m b e /\ Brick n m e e'.
Definition Valid_nm (n m : nat)(b : boundary) :=
 Boundary_i n m b (e'_nm n m b) /\ Boundary_j n m b (e_nm n m b) /\
 Brick n m (e_nm n m b) (e'_nm n m b).
```

Figure 9: Definitions of validities

```
Lemma P22_Valid_nm : forall b : boundary, Valid_nm 2 2 b.
Lemma Valid_nm_ind_2m : forall (b : boundary)(m : nat),
 2 <= m -> (forall b' : boundary, Valid_nm 2 m b') -> Valid_nm 2 (S m) b.
Lemma P2m_Valid_nm : forall (b : boundary)(m : nat), 2 <= m -> Valid_nm 2 m b.
Lemma Valid_nm_ind_nm : forall (b : boundary)(n m : nat),
 2 <= n -> 2 <= m -> (forall b' : boundary, Valid_nm n m b') ->
 Valid_nm (S n) m b.
Theorem e_nm_Valid : forall (b : boundary)(n m : nat),
 2 <= n -> 2 <= m -> Valid_nm n m b.

Theorem Pnm_Tileable : forall (b : boundary)(n m : nat),
 2 <= n -> 2 <= m -> exists (e e' : edge), Valid n m b e e'.
Proof.
  move => b n m H0 H1.
  exists (e_nm n m b).
  exists (e'_nm n m b).
  apply (e_nm_Valid b n m H0 H1).
Qed.
```

Figure 10: Main theorems in Coq

We first prove that the result is valid for all 2×2 boundaries. In a 2×2 problem, there are $|C|^8$ patterns of boundary colorings. Since $|C|$ is not bounded, there are infinitely many cases. Focusing on an equality of colors between facing edges, we can divide those patterns to 16 cases. We do not distinguish between 16 cases, we reduce the problem to 13 cases because there are 4 cases that can be proved in the same way. The validity of our algorithm **(Step 3)** is showed in `P22_Valid_nm`. Second, we show that the result is valid for all $2 \times m$ boundaries using an induction on m. The validity of our algorithm **(Step 2)** is showed in `P2m_Valid_nm`. Finally, we show that the result is valid for all $n \times m$ boundaries using an induction on n. The validity of our algorithm **(Step 1)** is showed in `e_nm_Valid`. The associated lemmas and theorem are listed in Figure 10.

The validity is proved in Coq as `e_nm_Valid`. Since we have implemented functions `e_nm` and `e'_nm`, only we need to show is to check a condition `Valid` for any input b boundary coloring. Theorem 1 about the tileablity is translated into a theorem `Pnm_Tileable` in Figure 10. We note that Theorem 1 is a theorem for an existence, but we always show a concrete valid tiling for proving the existence. All source codes and proofs are in the github repository showed in Appendix.

5 Conclusion

We formalized the notion of Wang tiles in Coq and proved a type of domino problem for a rectangular region with a boundary condition and a given brick Wang tile set. We extracted from this proof a Coq program to construct a valid tiling for a given rectangular region and a boundary condition.The generalization of the proof and its formalization to arbitrary bounded regions of the plane constitutes an interesting development of the present study. In particular, an interesting point is the relaxation of the inclusion of a 2×2 square and the possibility of tiling a non-rectangular region of the plane that does not contain one.

Acknowledgement

This work was partially supported by the Core Research for Evolutional Science and Technology (CREST) Program titled “Mathematics for Expressive Image Synthesis” of the Japan Science and Technology Agency (JST).

References

[1] The Coq Proof Assistant, https://coq.inria.fr/.

[2] R. Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 66:72, 1966.

[3] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang Tiles for Image and Texture Generation. *ACM Transaction on Graphics*, 22(3):287–294, 2003.

[4] K. Culik. An aperiodic set of 13 Wang tiles. *Discrete Mathematics*, 160(1-3):245–251, 1996.

[5] A. Derouet-Jourdan, Y. Mizoguchi, and M. Salvati. Wang Tiles Modeling of Wall Patterns. In *Symposium on Mathematical Progress in Expressive Image Synthesis (MEIS2015)*, volume 64 of *MI Lecture Note Series*, pages 61–70. Kyushu University, 2015.

[6] E. Jeandel and M. Rao. An aperiodic set of 11 Wang tiles. http://arxiv.org/abs/1506.06492, 2015.

[7] J. Kari. A small aperiodic set of Wang tiles. *Discrete Mathematics*, 160(1-3):259–264, 1996.

[8] J. Kopf, D. Cohen, O. Deussen, and D. Lischinski. Recursive Wang Tiles for Real-Time Blue Noise. *ACM Transaction on Graphics*, 25(3):509–518, 2006.

[9] J. Stam. Aperiodic texture mapping. Technical Report R046, European Research Consortium for Informatics and Mathematics (ERCIM), 1997.

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

Appendix

All source codes and proofs are in the github repository

https://github.com/KyushuUniversityMathematics/BrickCornerWangTiling.

To check examples following instructions.

1. To extract functions `e_nm` and `e'_nm`.
   `coqc TilingProgram.v`
   There exists an OCaml file `TilingProgam.ml` that contains examples of boundary colorings, such as `boundary44a` and `boundary44c`.
2. To execute rendering example file.
   `ocamlc TilingProgram.ml Tilingrender.ml -o TilingRender`
   There exist rendered images such as `b44a.svg` and `b44c.svg`.
3. Some external program such as `rsvg-convert` may be used to convert the image format from `svg` to `png`.
   eg. `svg-convert b44a.svg > b44a.png`

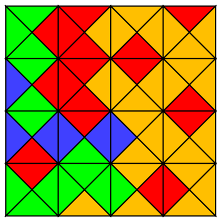


Figure 11: Example of an execution result ($n = m = 4$).