

Chapter 1

Library TilingProgram

1.1 Preference

Require Import *ssreflect ssrnat*.

Coercion *istrue* (*b* : *bool*) := *is_true* *b*.

2 つの自然数が等しければ 0 を, 異なれば 1 を返す関数.

Fixpoint *eq_to_bin* (*n m* : *nat*) : *nat* :=

match *n, m* with

| *O, O* ⇒ 0

| *O, S m'* ⇒ 1

| *S n', O* ⇒ 1

| *S n', S m'* ⇒ *eq_to_bin n' m'*

end.

Lemma *eq_to_bin_iff* : $\forall (n\ m : \text{nat}), n = m \leftrightarrow 0 = \text{eq_to_bin } n\ m$.

Proof.

induction *n*.

induction *m*.

split; move ⇒ *H*.

by [compute].

by [].

split; move ⇒ *H*.

discriminate *H*.

compute in *H*.

discriminate *H*.

induction *m*.

split; move ⇒ *H*.

discriminate *H*.

compute in *H*.

discriminate *H*.

```

simpl.
split; move => H.
apply IHn.
by [injection H].
apply IHn in H.
by [rewrite H].
Qed.

```

Lemma *eq_to_bin_nn* : $\forall n : \text{nat}, \text{eq_to_bin } n \ n = 0$.

Proof.

move => n.

apply *Logic.eq_sym*.

by [rewrite -eq_to_bin_iff].

Qed.

2 つの自然数が等しければ p を, 異なれば q を返す関数.

Definition *eq_to_if* ($n \ m \ p \ q : \text{nat}$) : $\text{nat} :=$

match *eq_to_bin* $n \ m$ with

| 0 => p

| _ => q

end.

特定の $C0$ 以外なら何でもよい場合は, $C_other2 \ C0$ で他の色を求める.

Definition *C_other2* ($C0 : \text{nat}$) : $\text{nat} :=$

match $C0$ with

| 0 => 1

| _ => 0

end.

Lemma *C_other2_neq* : $\forall C0 : \text{nat}, C0 \neq C_other2 \ C0$.

Proof.

induction $C0$.

by [simpl].

by [simpl].

Qed.

3 色以上使える環境で, 色 $C0$ と $C1$ が指定されたとき, それらと異なる色を返す関数.

Definition *C_other3* ($C0 \ C1 : \text{nat}$) : $\text{nat} :=$

match $C0$ with

| 0 => match $C1$ with

| 0 => 1

| 1 => 2

| _ => 1

end

| 1 => match $C1$ with

```

      | 0 ⇒ 2
      | _ ⇒ 0
    end
  | _ ⇒ match C1 with
      | 0 ⇒ 1
      | _ ⇒ 0
    end
end.

Lemma C_other3_neq :
  ∀ (C0 C1 : nat), C0 ≠ C_other3 C0 C1 ∧ C1 ≠ C_other3 C0 C1.
Proof.
  induction C0.
  induction C1.
  by [simpl].
  induction C1.
  by [simpl].
  by [simpl].
  induction C0.
  induction C1.
  by [simpl].
  by [simpl].
  induction C1.
  by [simpl].
  by [simpl].
Qed.

```

1.2 Wang tiling

境界条件とエッジ関数は、ともに “ x 座標と y 座標から色を返す関数” である。

Definition *boundary* := $\text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$.

Definition *edge* := $\text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$.

テスト用にプログラムを用いた Tiling を表示する関数も作ってみる。

Definition *null* { $A : \text{Type}$ } ($x : A$): A .

Proof.

apply x .

Qed.

Notation " \wedge " := (*null* 0).

Notation " $\#$ " := (*null* 1).

Open Scope *list_scope*.

Fixpoint *e_i* ($j : \text{nat}$) : $\text{edge} \rightarrow \text{nat} \rightarrow \text{list nat} :=$

fun ($e : \text{edge}$)($i : \text{nat}$) ⇒

```

match j with
| 0 ⇒ ^ :: nil
| S j' ⇒ (e_i j' e i) ++ ((e i (S j')) :: ^ :: nil)
end.
Fixpoint e'_i (j : nat) : edge → nat → list nat :=
fun (e' : edge)(i : nat) ⇒
match j with
| 0 ⇒ (e' i 0) :: nil
| S j' ⇒ (e'_i j' e' i) ++ (# :: (e' i (S j')) :: nil)
end.
Fixpoint e_e' (n m : nat)(e e' : edge) : list (list nat) :=
match n with
| 0 ⇒ (e_i m e 0) :: nil
| S n' ⇒ (e_e' n' m e e') ++ ((e'_i m e' (S n')) :: (e_i m e (S n')) :: nil)
end.
Definition tiling (n m : nat)(b : boundary)(e_ e'_ : boundary → edge) := e_e' n m (e_
b) (e'_ b).
  長方形サイズ  $n \times m$ , 境界条件 b, Tiling 関数 e_, e'_ から実際の Tiling を求める関数
  まずは  $P_{12}$  を Tiling する関数から. e は横エッジ用, e' は縦エッジ用.
Definition e_12 (b : boundary) : edge.
  横エッジはそのまま,  $e\ 0\ j = b\ 0\ j$ ,  $e\ 1\ j = b\ 2\ j$  とすればよい
rewrite /edge.
apply (fun i j : nat ⇒
match i with
| 0 ⇒ b 0 j
| _ ⇒ b 2 j
end).
Defined.
Definition e'_12 (b : boundary) : edge.
   $e'\ 1\ 0 = b\ 1\ 0$ ,  $e'\ 1\ 2 = b\ 1\ 3$  なので, j で induction
rewrite /edge.
apply (fun i j : nat ⇒
match j with
| 0 ⇒ b 1 0
| 1 ⇒ eq_to_if (b 1 0) (b 1 3)
      (eq_to_if (b 0 1) (b 2 1) (C_other2 (b 1 0)) (b 1 0))
      (eq_to_if (b 0 1) (b 2 1)
        (eq_to_if (b 0 2) (b 2 2) (C_other3 (b 1 0) (b 1 3)) (b 1 3))
        (b 1 0))
| _ ⇒ b 1 3
end).

```

Defined.

次に P_{22} を Tiling する関数.

Definition e_22 ($b : boundary$) : $edge$.

rewrite / $edge$.

```
apply (fun i j : nat =>
match i with
| 0 => b 0 j
| 1 => eq_to_if (b 1 0) (b 1 3)
      (eq_to_if (b 2 0) (b 2 3)
        (C_other3 (b 0 j) (b 3 j))
        (eq_to_if (b 0 1) (b 3 1)
          (b 0 j)
          (match j with
            | 0 => C_other2 (b 0 0)
            | 1 => b 3 1
            | _ => C_other2 (b 0 j)
          end)))
      (eq_to_if (b 2 0) (b 2 3)
        (eq_to_if (b 0 1) (b 3 1)
          (b 3 j)
          (match j with
            | 0 => C_other2 (b 3 0)
            | 1 => b 0 1
            | _ => C_other2 (b 3 j)
          end)))
      (match j with
        | 0 => b 3 2
        | 1 => b 0 1
        | _ => b 3 2
      end))
| _ => b 3 j
end).
```

Defined.

Definition e'_22 ($b : boundary$) : $edge$.

上で定義した e_22 に基づいて定義する

rewrite / $edge$.

```
apply (fun i j : nat =>
match i with
| 0 => 0
| 1 => (e'_12 (fun i j : nat =>
      match i with
```

```

      | 0 ⇒ b 0 j
      | 1 ⇒ b 1 j
      | _ ⇒ (e_22 b 1 j)
    end) 1 j)
| _ ⇒ (e'_12 (fun i j : nat ⇒
      match i with
      | 0 ⇒ (e_22 b 1 j)
      | _ ⇒ b (S i) j
      end) 1 j)
end
).
Defined.

```

$P_{(n+1)m}$ の境界条件を, P_{nm} と P_{1m} に分割し, 前者の境界条件を出す関数.

Definition *bSnm_to_bnm* ($m : nat$) : *boundary* → *boundary*.

```

move ⇒ b.
rewrite /boundary.
apply (fun i j : nat ⇒
match m with
| 0 ⇒ b i j
| 1 ⇒ b i j
| _ ⇒ match i with
      | 0 ⇒ match j with
          | 0 ⇒ 0
          | 1 ⇒ b 0 1
          | 2 ⇒ b 0 2
          | _ ⇒ C_other2 (b 0 j)
        end
      | _ ⇒ b (S i) j
    end)
end).
Defined.

```

$P_{(n+1)m}$ の境界条件を, P_{nm} と P_{1m} に分割し, 後者の境界条件を出す関数.

Definition *bSnm_to_b1m* ($m : nat$) : *boundary* → *boundary*.

```

move ⇒ b.
rewrite /boundary.
apply (fun i j : nat ⇒
match m with
| 0 ⇒ b i j
| 1 ⇒ b i j
| _ ⇒ match i with
      | 0 ⇒ b 0 j

```

```

      | 1 ⇒ b 1 j
      | - ⇒ match j with
        | 0 ⇒ 0
        | 1 ⇒ b 0 1
        | 2 ⇒ b 0 2
        | - ⇒ C_other2 (b 0 j)
      end
    end
  end
end

```

end).

Defined.

bSnm_to_b1m で出てくる P_{1m} を Tiling する関数.

Definition $e_{1m} (b : \text{boundary}) : \text{edge}$.

横エッジはそのまま, $e_0 j = b_0 j$, $e_1 j = b_2 j$ とすればよい

rewrite /edge.

```

apply (fun i j : nat ⇒
match i with
| 0 ⇒ b 0 j
| - ⇒ b 2 j
end).

```

Defined.

Definition $e'_{1m} (m : \text{nat})(b : \text{boundary}) : \text{edge}$.

縦エッジは, $b_1 0 = e'_1 0 <> e'_1 1 <> e'_1 2 = \dots = e'_1 m = b_1 (S m)$ にする

rewrite /edge.

```

apply (fun i j : nat ⇒
match i with
| 0 ⇒ 0
| - ⇒ match j with
      | 0 ⇒ b 1 0
      | 1 ⇒ C_other3 (b 1 0) (b 1 (S m))
      | - ⇒ b 1 (S m)
    end
end).

```

end).

Defined.

P_{n2} を Tiling する関数.

Fixpoint $e_{n2} (n : \text{nat}) : \text{boundary} \rightarrow \text{edge} :=$

```

fun b : boundary ⇒
match n with
| 0 | 1 ⇒ e_12 b
| 2 ⇒ e_22 b
| S n' ⇒ fun (i j : nat) ⇒

```

```

      match i with
      | 0 ⇒ (bSnm_to_b1m 2 b) 0 j
      | S i' ⇒ e_n2 n' (bSnm_to_bnm 2 b) i' j
      end
    end.

  end.

Fixpoint e'_n2 (n : nat) : boundary → edge :=
  fun b : boundary ⇒
  match n with
  | 0 | 1 ⇒ e'_12 b
  | 2 ⇒ e'_22 b
  | S n' ⇒ fun (i j : nat) ⇒
      match i with
      | 0 ⇒ 0
      | 1 ⇒ e'_1m 2 (bSnm_to_b1m 2 b) 1 j
      | S i' ⇒ e'_n2 n' (bSnm_to_bnm 2 b) i' j
      end
    end.

  end.

```

P_{nm} での境界条件および Tiling 関数を P_{mn} のものに置き換える関数. やっていることはただの引数順序の入れ替え. 横エッジ e と縦エッジ e' も入れ替える.

```

Definition bnm_to_bmn (b : boundary) : boundary.
move ⇒ i j.
apply (b j i).
Defined.

```

```

Definition enm_to_emn (e : boundary → edge) : boundary → edge.
move ⇒ b i j.
apply (e (bnm_to_bmn b) j i).
Defined.

```

3 色以上, 2×2 以上のときに, P_{nm} を Tiling する関数.

```

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.

  end.

Fixpoint e'_nm (n m : nat) : boundary → edge :=
  fun b : boundary ⇒

```



```

match n with
| 0 | 1  $\Rightarrow$  e'_1m m b
| 2  $\Rightarrow$  enm_to_emn (fun b'  $\Rightarrow$  e_n2 m b') b
| S n'  $\Rightarrow$  fun (i j : nat)  $\Rightarrow$ 
      match i with
      | 0  $\Rightarrow$  0
      | 1  $\Rightarrow$  e'_1m m (bSnm_to_b1m m b) 1 j
      | S i'  $\Rightarrow$  e'_nm n' m (bSnm_to_bnm m b) i' j
      end
end.

```

Tiling 関数を e_nm, e'_nm に固定したものを定義.

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

1.3 Examples

Compute (*eq_to_bin* 8 8).

```

= 0
: nat

```

Compute (*eq_to_bin* 4 7).

```

= 1
: nat

```

Compute (*C_other2* 1).

```

= 0
: nat

```

Compute (*C_other3* 0 1).

```

= 2
: nat

```

Compute (*C_other3* 2 0).

```

= 1
: nat

```

Compute (*tiling* 1 2 (fun _ _ \Rightarrow 0) e_12 e'_12).

```

= (^ :: 0 :: ^ :: 0 :: ^ :: nil)
:: (0 :: # :: 1 :: # :: 0 :: nil)
:: (^ :: 0 :: ^ :: 0 :: ^ :: nil) :: nil
: list (list nat)

```

がタイルを表す. つまり # の上下左右が Brick Corner Wang Tiling の条件を満たせばよい.

Compute (*tiling* 1 2 (fun _ j \Rightarrow match *j* with 1 \Rightarrow 2 | _ \Rightarrow 1 end) *e*₋₁₂ *e'*₋₁₂).

```

= (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (1 :: # :: 0 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 2 2 (fun _ _ \Rightarrow 0) *e*₋₂₂ *e'*₋₂₂).

```

= (^ :: 0 :: ^ :: 0 :: ^ :: nil)
:: (0 :: # :: 0 :: # :: 0 :: nil)
:: (^ :: 1 :: ^ :: 1 :: ^ :: nil)
:: (0 :: # :: 0 :: # :: 0 :: nil)
:: (^ :: 0 :: ^ :: 0 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 2 2 (fun *i j* \Rightarrow match *i* with 0 \Rightarrow 2 | _ \Rightarrow match *j* with 0 \Rightarrow 0 | _ \Rightarrow 1 end end) *e*₋₂₂ *e'*₋₂₂).

```

= (^ :: 2 :: ^ :: 2 :: ^ :: nil)
:: (0 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (0 :: # :: 0 :: # :: 1 :: nil)
:: (^ :: 1 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 2 2 (fun *i j* \Rightarrow match *j* with 1 \Rightarrow 2 | 3 \Rightarrow 1 | _ \Rightarrow match *i* with 1 \Rightarrow 0 | _ \Rightarrow 1 end end) *e*₋₂₂ *e'*₋₂₂).

```

= (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (0 :: # :: 2 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (1 :: # :: 0 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 1 4 (*bSnm_to_b1m* 4 (fun *i j* \Rightarrow match *i* with 0 \Rightarrow 2 | _ \Rightarrow match *j* with 0 \Rightarrow 0 | _ \Rightarrow 1 end end)) *e*_{-1m} (*e'*_{-1m} 4)).

```

= (^ :: 2 :: ^ :: 2 :: ^ :: 2 :: ^ :: 2 :: ^ :: nil)
:: (0 :: # :: 2 :: # :: 1 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 2 :: ^ :: 0 :: ^ :: 0 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 1 4 (*bSnm_to_b1m* 4 (fun *i j* ⇒ match *j* with 1 ⇒ 2 | 3 ⇒ 1 | _ ⇒ match *i* with 1 ⇒ 0 | _ ⇒ 1 end end)) *e_1m* (*e'_1m* 4)).

```

= (^ :: 2 :: ^ :: 1 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil)
:: (0 :: # :: 1 :: # :: 0 :: # :: 0 :: # :: 0 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: 0 :: ^ :: 0 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 3 2 (fun *i j* ⇒ match *i* with 0 ⇒ 2 | _ ⇒ match *j* with 0 ⇒ 0 | _ ⇒ 1 end end) (*e_n2* 3) (*e'_n2* 3)).

```

= (^ :: 2 :: ^ :: 2 :: ^ :: nil)
:: (0 :: # :: 2 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 2 :: ^ :: nil)
:: (0 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (0 :: # :: 0 :: # :: 1 :: nil)
:: (^ :: 1 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 4 2 (fun *i j* ⇒ match *j* with 1 ⇒ 2 | 3 ⇒ 1 | _ ⇒ match *i* with 1 ⇒ 0 | _ ⇒ 1 end end) (*e_n2* 4) (*e'_n2* 4)).

```

= (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (0 :: # :: 2 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (1 :: # :: 0 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil)
:: (1 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 0 :: ^ :: 0 :: ^ :: nil)
:: (1 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 2 4 (*bnm_to_bmn* (fun *i j* ⇒ match *i* with 0 ⇒ 2 | _ ⇒ match *j* with 0 ⇒ 0 | _ ⇒ 1 end end)) (*enm_to_emn* (*e'_n2* 4)) (*enm_to_emn* (*e_n2* 4))).

```

= (^ :: 0 :: ^ :: 0 :: ^ :: 0 :: ^ :: 0 :: ^ :: nil)
:: (2 :: # :: 2 :: # :: 2 :: # :: 2 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 2 :: ^ :: 1 :: ^ :: 0 :: ^ :: nil)
:: (2 :: # :: 2 :: # :: 2 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 1 :: ^ :: 1 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling* 2 4 (*bnm_to_bmn* (fun *i j* ⇒ match *j* with 1 ⇒ 2 | 3 ⇒ 1 | _ ⇒ match *i* with 1 ⇒ 0 | _ ⇒ 1 end end)) (*enm_to_emn* (*e'_n2* 4)) (*enm_to_emn* (*e_n2* 4))).

```

= (^ :: 0 :: ^ :: 1 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil)
:: (2 :: # :: 2 :: # :: 2 :: # :: 0 :: # :: 2 :: nil)
:: (^ :: 2 :: ^ :: 0 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil)
:: (1 :: # :: 1 :: # :: 1 :: # :: 0 :: # :: 1 :: nil)
:: (^ :: 1 :: ^ :: 1 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling_nm* 4 4 (fun *i j* ⇒ match *i* with 0 ⇒ 2 | 3 ⇒ match *j* with 0 ⇒ 5 | _ ⇒ 1 end | _ ⇒ match *j* with 1 ⇒ 3 | _ ⇒ 4 end end)).

```

= (^ :: 2 :: ^ :: 2 :: ^ :: 2 :: ^ :: 2 :: ^ :: nil)
:: (4 :: # :: 0 :: # :: 4 :: # :: 4 :: # :: 4 :: nil)
:: (^ :: 2 :: ^ :: 2 :: ^ :: 0 :: ^ :: 0 :: ^ :: nil)
:: (4 :: # :: 0 :: # :: 4 :: # :: 4 :: # :: 4 :: nil)
:: (^ :: 2 :: ^ :: 2 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil)
:: (5 :: # :: 5 :: # :: 5 :: # :: 5 :: # :: 1 :: nil)
:: (^ :: 0 :: ^ :: 0 :: ^ :: 0 :: ^ :: 1 :: ^ :: nil)
:: (4 :: # :: 4 :: # :: 4 :: # :: 4 :: # :: 4 :: nil)
:: (^ :: 3 :: ^ :: 4 :: ^ :: 4 :: ^ :: 4 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling_nm* 4 4 (fun *i j* ⇒ match *j* with 0 ⇒ match *i* with 2 | 3 ⇒ 3 | _ ⇒ 4 end | 1 ⇒ 2 | 3 ⇒ 1 | _ ⇒ match *i* with 0 ⇒ 0 | _ ⇒ 5 end end)).

```

= (^ :: 2 :: ^ :: 0 :: ^ :: 1 :: ^ :: 0 :: ^ :: nil)
:: (4 :: # :: 0 :: # :: 5 :: # :: 5 :: # :: 5 :: nil)
:: (^ :: 2 :: ^ :: 0 :: ^ :: 0 :: ^ :: 1 :: ^ :: nil)
:: (3 :: # :: 0 :: # :: 5 :: # :: 5 :: # :: 5 :: nil)
:: (^ :: 2 :: ^ :: 0 :: ^ :: 1 :: ^ :: 0 :: ^ :: nil)
:: (3 :: # :: 3 :: # :: 3 :: # :: 5 :: # :: 5 :: nil)
:: (^ :: 0 :: ^ :: 1 :: ^ :: 1 :: ^ :: 5 :: ^ :: nil)
:: (4 :: # :: 4 :: # :: 4 :: # :: 0 :: # :: 5 :: nil)
:: (^ :: 2 :: ^ :: 5 :: ^ :: 1 :: ^ :: 5 :: ^ :: nil) :: nil
: list (list nat)

```

Compute (*tiling_nm* 4 4 (fun *i j* \Rightarrow match *j* with 0 \Rightarrow match *i* with 2 | 3 \Rightarrow 3 | _ \Rightarrow 2 end
 | 1 \Rightarrow 2 | 3 \Rightarrow 1 | _ \Rightarrow match *i* with 0 \Rightarrow 0 | _ \Rightarrow 1 end end)).

```
= (^ :: 2 :: ^ :: 0 :: ^ :: 1 :: ^ :: 0 :: ^ :: nil)
:: (2 :: # :: 0 :: # :: 1 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 0 :: ^ :: 0 :: ^ :: 1 :: ^ :: nil)
:: (3 :: # :: 0 :: # :: 1 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 0 :: ^ :: 1 :: ^ :: 0 :: ^ :: nil)
:: (3 :: # :: 3 :: # :: 3 :: # :: 1 :: # :: 1 :: nil)
:: (^ :: 0 :: ^ :: 2 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil)
:: (2 :: # :: 2 :: # :: 2 :: # :: 0 :: # :: 1 :: nil)
:: (^ :: 2 :: ^ :: 1 :: ^ :: 1 :: ^ :: 1 :: ^ :: nil) :: nil
: list (list nat)
```

1.4 Main theorems

WangTiling.v から, “Tiling 可能” の定義をインポート. 以下の 3 つを全て同時に満たせば Tiling できていることになる.

Definition *Boundary_i* (*n m* : nat)(*b* : boundary)(*e'* : edge) :=
 $\forall i : \text{nat}, e' \ i \ 0 = b \ i \ 0 \wedge e' \ i \ m = b \ i \ (S \ m) \vee i = 0 \vee n < i.$

Definition *Boundary_j* (*n m* : nat)(*b* : boundary)(*e* : edge) :=
 $\forall j : \text{nat}, e \ 0 \ j = b \ 0 \ j \wedge e \ n \ j = b \ (S \ n) \ j \vee j = 0 \vee m < j.$

Definition *Brick* (*n m* : nat)(*e e'* : edge) :=
 $\forall i \ j : \text{nat},$
 $(e \ i \ (S \ j) = e \ (S \ i) \ (S \ j) \wedge e' \ (S \ i) \ j \neq e' \ (S \ i) \ (S \ j)) \vee$
 $(e \ i \ (S \ j) \neq e \ (S \ i) \ (S \ j) \wedge e' \ (S \ i) \ j = e' \ (S \ i) \ (S \ j)) \vee$
 $n \leq i \vee m \leq j.$

Definition *Tileable* (*n m* : nat)(*b* : boundary)(*e e'* : edge) :=
 $\text{Boundary_i } n \ m \ b \ e' \wedge \text{Boundary_j } n \ m \ b \ e \wedge \text{Brick } n \ m \ e \ e'.$

Definition *Tileable_nm* (*n m* : nat)(*b* : boundary) :=
 $\text{Boundary_i } n \ m \ b \ (e_nm \ n \ m \ b) \wedge \text{Boundary_j } n \ m \ b \ (e_nm \ n \ m \ b) \wedge$
 $\text{Brick } n \ m \ (e_nm \ n \ m \ b) \ (e_nm \ n \ m \ b).$

P_{22} は 3 色以上で Tileable という補題.

Lemma *P22_Tileable_nm* : $\forall b : \text{boundary}, \text{Tileable_nm } 2 \ 2 \ b.$

Proof.

move $\Rightarrow b$.

repeat split.

induction *i*.

by [right; left].

induction *i*.

left.

```

split.
by [compute].
by [compute].
induction  $i$ .
left.
split.
by [compute].
by [compute].
by [repeat right].
induction  $j$ .
by [right; left].
induction  $j$ .
left.
split.
by [compute].
by [compute].
induction  $j$ .
left.
split.
by [compute].
by [compute].
by [repeat right].
rewrite / $e'_{-nm}/e_{-nm}/enm\_to\_emn/bnm\_to\_bmn/e'_{-n2}/e_{-n2}/e'_{-22}/e_{-22}/e'_{-12}/eq\_to\_if$ /.
simpl.
move  $\Rightarrow i\ j$ .
simpl.
induction  $j$ .
induction  $i$ .
remember (eq_to_bin (b 0 1) (b 3 1)).
induction  $n$ .
induction (eq_to_bin (b 0 2) (b 3 2)).
remember (eq_to_bin (b 1 0) (C_other3 (b 1 0) (b 1 3))).
induction  $n$ .
apply eq_to_bin_iff in Heqn0.
right; left.
split.
apply C_other2_neq.
apply Heqn0.
simpl.
left.
split.
by [].

```

```

apply C_other3_neq.
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite eq_to_bin_nn.
right; left.
split.
apply C_other2_neq.
by [].
rewrite -Heqn0.
left.
split.
by [].
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn0.
discriminate Heqn0.
induction (eq_to_bin (b 0 2) (b 3 2)).
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite -Heqn0.
apply eq_to_bin_iff in Heqn0.
rewrite Heqn0.
induction (eq_to_bin (b 2 0) (b 2 3)).
right; left.
split.
apply C_other3_neq.
by [].
right; left.
split.
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn.
discriminate Heqn.
by [].
rewrite eq_to_bin_nn.
induction (eq_to_bin (b 2 0) (C_other2 (b 2 3))).
right; left.
split.
apply C_other3_neq.
by [].
right; left.
split.
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn.

```

```

discriminate Heqn.
by [].
rewrite eq_to_bin_nn.
induction (eq_to_bin (b 2 0) (b 2 3)).
right; left.
split.
apply C_other3_neq.
by [].
right; left.
split.
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn.
discriminate Heqn.
by [].
induction i.
remember (eq_to_bin (b 0 1) (b 3 1)).
induction n.
apply eq_to_bin_iff in Heqn.
rewrite Heqn.
induction (eq_to_bin (b 0 2) (b 3 2)).
remember (eq_to_bin (b 1 0) (C_other3 (b 1 0) (b 1 3))).
induction n.
apply eq_to_bin_iff in Heqn0.
apply False_ind.
elim (C_other3_neq (b 1 0) (b 1 3))  $\Rightarrow$  H H0.
apply (H Heqn0).
left.
split.
by [].
apply C_other3_neq.
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite eq_to_bin_nn.
right; left.
split.
apply not_eq_sym.
apply C_other2_neq.
by [].
rewrite -Heqn0.
left.
split.
by [].

```



```

apply C_other2_neq.
induction (eq_to_bin (b 0 2) (b 3 2)).
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite -Heqn0.
remember (eq_to_bin (b 2 0) (b 2 3)).
induction n0.
apply eq_to_bin_iff in Heqn1.
rewrite Heqn1.
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
by [].
left.
split.
by [].
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn1.
discriminate Heqn1.
rewrite eq_to_bin_nn.
remember (eq_to_bin (b 2 0) (C_other2 (b 2 3))).
induction n1.
apply eq_to_bin_iff in Heqn1.
rewrite Heqn1.
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
by [].
left.
split.
by [].
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn1.
discriminate Heqn1.
rewrite eq_to_bin_nn.
remember (eq_to_bin (b 2 0) (C_other2 (b 2 3))).
induction n1.
apply eq_to_bin_iff in Heqn1.
rewrite Heqn1.
remember (eq_to_bin (C_other2 (b 2 3)) (b 2 3)).

```

```

induction n1.
apply eq_to_bin_iff in Heqn0.
rewrite Heqn0.
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
by [].
left.
split.
by [].
apply not_eq_sym.
apply C_other2_neq.
remember (eq_to_bin (b 2 0) (b 2 3)).
induction n2.
apply eq_to_bin_iff in Heqn2.
rewrite Heqn2.
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
by [].
left.
split.
by [].
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn2.
discriminate Heqn2.
by [right; right; left].
induction i.
remember (eq_to_bin (b 0 2) (b 3 2)).
induction n.
induction (eq_to_bin (b 0 1) (b 3 1)).
remember (eq_to_bin (b 1 0) (C_other3 (b 1 0) (b 1 3))).
induction n.
apply eq_to_bin_iff in Heqn0.
elim (C_other3_neq (b 1 0) (b 1 3))  $\Rightarrow$  H H0.
apply False_ind.
apply (H Heqn0).
remember (eq_to_bin (C_other3 (b 1 0) (b 1 3)) (b 1 3)).
induction n0.
apply eq_to_bin_iff in Heqn1.

```

```

elim (C_other3_neq (b 1 0) (b 1 3))  $\Rightarrow$  H H0.
rewrite Heqn1 in H0.
apply False_ind.
by [apply H0].
induction j.
left.
split.
by [].
apply not_eq_sym.
apply C_other3_neq.
by [repeat right].
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite eq_to_bin_nn.
right; left.
split.
apply C_other2_neq.
induction j.
by [].
by [].
rewrite -Heqn0.
induction j.
left.
split.
by [].
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn0.
discriminate Heqn0.
by [repeat right].
induction (eq_to_bin (b 0 1) (b 3 1)).
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite -Heqn0.
apply eq_to_bin_iff in Heqn0.
rewrite Heqn0.
induction (eq_to_bin (b 2 0) (b 2 3)).
induction j.
right; left.
split.
apply C_other3_neq.
by [].
by [repeat right].

```

```

induction j.
right; left.
split.
move  $\Rightarrow H$ .
rewrite  $H$   $eq\_to\_bin\_nn$  in  $Hegn$ .
discriminate  $Hegn$ .
by [].
by [repeat right].
rewrite  $eq\_to\_bin\_nn$ .
induction ( $eq\_to\_bin$  ( $C\_other2$  ( $b$  2 0)) ( $b$  2 3)).
right; left.
split.
apply  $C\_other3\_neq$ .
induction j.
by [].
by [].
right; left.
split.
move  $\Rightarrow H$ .
rewrite  $H$   $eq\_to\_bin\_nn$  in  $Hegn$ .
discriminate  $Hegn$ .
induction j.
by [].
by [].
remember ( $eq\_to\_bin$  ( $b$  1 0) ( $b$  1 3)).
induction  $n1$ .
apply  $eq\_to\_bin\_iff$  in  $Hegn1$ .
rewrite  $Hegn1$ .
induction ( $eq\_to\_bin$  ( $b$  2 3) ( $b$  2 3)).
right; left.
split.
apply  $C\_other3\_neq$ .
induction j.
by [].
by [].
right; left.
split.
move  $\Rightarrow H$ .
rewrite  $H$   $eq\_to\_bin\_nn$  in  $Hegn$ .
discriminate  $Hegn$ .
induction j.
by [].

```

```

by [].
induction j.
left.
split.
by [].
move  $\Rightarrow H$ .
rewrite  $H$  eq_to_bin_nn in Heqn1.
discriminate Heqn1.
by [repeat right].
induction i.
remember (eq_to_bin (b 0 2) (b 3 2)).
induction n.
apply eq_to_bin_iff in Heqn.
rewrite Heqn.
induction (eq_to_bin (b 0 1) (b 3 1)).
remember (eq_to_bin (C_other3 (b 1 0) (b 1 3)) (b 1 3)).
induction n.
apply eq_to_bin_iff in Heqn0.
elim (C_other3_neq (b 1 0) (b 1 3))  $\Rightarrow H$  H0.
rewrite Heqn0 in H0.
apply False_ind.
by [apply H0].
induction j.
left.
split.
by [].
apply not_eq_sym.
apply C_other3_neq.
by [repeat right].
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite eq_to_bin_nn.
right; left.
split.
apply not_eq_sym.
apply C_other2_neq.
induction j.
by [].
by [].
rewrite -Heqn0.
induction j.
left.

```

```

split.
by [].
apply not_eq_sym.
apply C_other2_neq.
by [repeat right].
induction (eq_to_bin (b 0 1) (b 3 1)).
remember (eq_to_bin (b 1 0) (b 1 3)).
induction n0.
rewrite -Heqn0.
remember (eq_to_bin (b 2 0) (b 2 3)).
induction n0.
apply eq_to_bin_iff in Heqn1.
rewrite Heqn1.
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
induction j.
by [].
by [].
induction j.
left.
split.
by [].
move  $\Rightarrow$  H.
rewrite H eq_to_bin_nn in Heqn1.
discriminate Heqn1.
by [repeat right].
rewrite eq_to_bin_nn.
remember (eq_to_bin (C_other2 (b 2 0)) (b 2 3)).
induction n1.
apply eq_to_bin_iff in Heqn1.
rewrite Heqn1.
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
induction j.
by [].
by [].
induction j.
left.

```

```

split.
by [].
move  $\Rightarrow H$ .
rewrite  $H$   $eq\_to\_bin\_nn$  in  $Hegn1$ .
discriminate  $Hegn1$ .
by [repeat right].
induction ( $eq\_to\_bin$  ( $b$  1 0) ( $b$  1 3)).
rewrite  $eq\_to\_bin\_nn$ .
right; left.
split.
apply  $not\_eq\_sym$ .
apply  $C\_other3\_neq$ .
induction  $j$ .
by [].
by [].
right; left.
split.
move  $\Rightarrow H$ .
rewrite  $H$   $eq\_to\_bin\_nn$  in  $Hegn$ .
discriminate  $Hegn$ .
induction  $j$ .
by [].
by [].
by [right; right; left].
Qed.

```

$m \geq 2$ で P_{2m} が Tileable なら, $P_{2(m+1)}$ も Tileable という補題.

Lemma $Tileable_nm_ind_2m : \forall (b : boundary)(m : nat),$
 $2 \leq m \rightarrow (\forall b' : boundary, Tileable_nm\ 2\ m\ b') \rightarrow Tileable_nm\ 2\ (S\ m)\ b.$

Proof.

move $\Rightarrow b\ m\ H\ H0$.

rewrite $/Tileable_nm/e_nm/e_nm$.

replace (fun $b' : boundary \Rightarrow e_n2\ m.+1\ b'$) with (fun $b' : boundary \Rightarrow$ (fun ($i\ j : nat$)
 \Rightarrow match i with 0 \Rightarrow ($bSnm_to_b1m\ 2\ b'$) 0 $j \mid S\ i' \Rightarrow e_n2\ m\ (bSnm_to_bnm\ 2\ b')$ $i'\ j$
end)).

Focus 2.

rewrite $/e_n2$.

induction m .

discriminate H .

induction m .

discriminate H .

by [].

replace (fun $b' : boundary \Rightarrow e_n2\ m.+1\ b'$) with (fun $b' : boundary \Rightarrow$ (fun ($i\ j :$

```

nat)  $\Rightarrow$  match  $i$  with 0  $\Rightarrow$  0 | 1  $\Rightarrow$   $e'_{1m} \ 2 \ (bSnm\_to\_b1m \ 2 \ b') \ 1 \ j \mid S \ i' \Rightarrow e'_{n2} \ m$ 
( $bSnm\_to\_bnm \ 2 \ b') \ i' \ j$  end)).
Focus 2.
rewrite / $e'_{n2}$ .
induction  $m$ .
discriminate  $H$ .
induction  $m$ .
discriminate  $H$ .
by [].
induction  $m$ .
discriminate  $H$ .
induction  $m$ .
discriminate  $H$ .
clear  $IHm \ IHm0$ .
move : ( $H0 \ (bnm\_to\_bmn \ (bSnm\_to\_bnm \ 2 \ (bnm\_to\_bmn \ b)))$ ).
rewrite / $Tileable\_nm / Boundary\_i / Boundary\_j / Brick / e'_{nm} / e_{nm} / enm\_to\_emn / bnm\_to\_bmn / bSnm$ .
elim  $\Rightarrow H1$ .
elim  $\Rightarrow H2 \ H3$ .
repeat split.
clear  $H2 \ H3$ .
move  $\Rightarrow i$ .
induction  $i$ .
by [right; left].
induction  $i$ .
left.
split.
by [].
case ( $H1 \ 1$ )  $\Rightarrow H2$ .
apply  $H2$ .
case  $H2$ ; discriminate.
induction  $i$ .
left.
split.
by [].
case ( $H1 \ 2$ )  $\Rightarrow H2$ .
apply  $H2$ .
case  $H2$ ; discriminate.
by [repeat right].
clear  $H1 \ H3$ .
move  $\Rightarrow j$ .
induction  $j$ .
by [right; left].

```



```

induction j.
rewrite /e'_1m.
by [left].
case (H2 (S j))  $\Rightarrow$  H3.
left.
apply H3.
case H3  $\Rightarrow$  H4.
discriminate H4.
repeat right.
apply H4.
move : H2 H3.
move  $\Rightarrow$  H2 H3 i j.
induction j.
induction i.
induction m.
rewrite /e_22.
right; left.
split.
apply C_other3_neq.
by [].
right; left.
split.
apply C_other3_neq.
by [].
induction i.
induction m.
rewrite /e_22.
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
by [].
right; left.
split.
apply not_eq_sym.
apply C_other3_neq.
by [].
by [right; right; left].
move : H3.
rewrite /e'_22/e_22/e'_12.
move  $\Rightarrow$  H3.
apply H3.

```

Qed.

$m \geq 2$ なら, P_{2m} は 3 色以上で Tileable という補題.

Lemma $P2m_Tileable_nm : \forall (b : boundary)(m : nat), 2 \leq m \rightarrow Tileable_nm \ 2 \ m \ b$.

Proof.

induction m .

discriminate.

clear IHm .

induction m .

discriminate.

clear IHm .

move : b .

induction m .

move $\Rightarrow b \ H$.

apply $P22_Tileable_nm$.

move $\Rightarrow b \ H$.

apply $Tileable_nm_ind_2m$.

apply H .

move $\Rightarrow b'$.

apply IHm .

apply H .

Qed.

$n, m \geq 2$ で P_{nm} が Tileable なら, $P_{(n+1)m}$ も Tileable という補題.

Lemma $Tileable_nm_ind_nm : \forall (b : boundary)(n \ m : nat),$

$2 \leq n \rightarrow 2 \leq m \rightarrow (\forall b' : boundary, Tileable_nm \ n \ m \ b') \rightarrow Tileable_nm \ (S \ n) \ m \ b$.

Proof.

move $\Rightarrow b \ n \ m \ H \ H0 \ H1$.

induction n .

discriminate H .

induction n .

discriminate H .

clear $IHn \ IHn0$.

move : $(H1 \ (bSnm_to_bnm \ m \ b))$.

rewrite $/Tileable_nm/ Boundary_i/ Boundary_j/ Brick$.

elim $\Rightarrow H2$.

elim $\Rightarrow H3 \ H4$.

repeat split.

clear $H3 \ H4$.

induction i .

by [right; left].

move : $(H2 \ i)$.

replace $(e'_nm \ n.+3 \ m \ b)$ with $(\text{fun } (i \ j : nat) \Rightarrow \text{match } i \text{ with } 0 \Rightarrow 0 \mid 1 \Rightarrow e'_1m \ m$

```

(bSnm_to_b1m m b) 1 j | S i'  $\Rightarrow$  e'_nm n.+2 m (bSnm_to_bnm m b) i' j end).
Focus 2.
by [rewrite /e'_nm].
induction m.
discriminate H0.
induction m.
discriminate H0.
clear IHm IHm0.
case  $\Rightarrow$  H3.
left.
rewrite /bSnm_to_bnm in H3.
rewrite /e'_1m/bSnm_to_b1m/bSnm_to_bnm.
induction i.
by [].
apply H3.
case H3  $\Rightarrow$  H4.
rewrite H4.
rewrite /e'_1m/bSnm_to_b1m/bSnm_to_bnm.
by [left].
repeat right.
apply H4.
clear H2 H4.
move  $\Rightarrow$  j.
move : (H3 j).
induction j.
by [right; left].
replace (e_nm n.+3 m.+2 b) with (fun (i j : nat)  $\Rightarrow$  match i with 0  $\Rightarrow$  (bSnm_to_b1m
m.+2 b) 0 j | S i'  $\Rightarrow$  e_nm n.+2 m.+2 (bSnm_to_bnm m.+2 b) i' j end).
Focus 2.
by [rewrite /e_nm].
rewrite /bSnm_to_bnm/bSnm_to_b1m.
induction m.
discriminate H0.
induction m.
discriminate H0.
clear IHm IHm0.
case  $\Rightarrow$  H4.
left.
split.
by [].
apply H4.
right.

```

```

apply H4.
clear H2.
replace (e'_nm n.+3 m b) with (fun (i j : nat) => match i with 0 => 0 | 1 => e'_1m m
(bSnm_to_b1m m b) 1 j | S i' => e'_nm n.+2 m (bSnm_to_bnm m b) i' j end).
Focus 2.
by [rewrite /e'_nm].
replace (e_nm n.+3 m b) with (fun (i j : nat) => match i with 0 => (bSnm_to_b1m m b)
0 j | S i' => e_nm n.+2 m (bSnm_to_bnm m b) i' j end).
Focus 2.
by [rewrite /e_nm].
move => i j.
induction i.
Focus 2.
apply H4.
case (H3 j.+1) => H5.
elim H5 => H6 H7.
rewrite H6.
rewrite /bSnm_to_b1m/bSnm_to_bnm/enm_to_emn/bnm_to_bmn/e'_1m.
induction m.
discriminate H0.
induction m.
discriminate H0.
clear IHm IHm0.
induction j.
left.
split.
by [].
apply C_other3_neq.
induction j.
left.
split.
by [].
apply not_eq_sym.
apply C_other3_neq.
right; left.
split.
apply C_other2_neq.
by [].
case H5 => H6.
discriminate H6.
repeat right.
apply H6.

```

Qed.

$n, m \geq 2$ なら, P_{nm} は 3 色以上で Tileable という補題.

Lemma $Pnm_Tileable_nm : \forall (b : boundary)(n m : nat),$
 $2 \leq n \rightarrow 2 \leq m \rightarrow Tileable_nm\ n\ m\ b.$

Proof.

move $\Rightarrow b\ n\ m\ H\ H0.$

induction $n.$

discriminate $H.$

induction $n.$

discriminate $H.$

clear $IHn\ IHn0.$

induction $m.$

discriminate $H0.$

induction $m.$

discriminate $H0.$

clear $IHm\ IHm0.$

move : $b.$

induction $n.$

move $\Rightarrow b.$

apply $(P2m_Tileable_nm _ _ H0).$

move $\Rightarrow b.$

apply $Tileable_nm_ind_nm.$

apply $H.$

apply $H0.$

apply $IHn.$

apply $H.$

Qed.

Theorem $Pnm_Tileable : \forall (b : boundary)(n m : nat),$
 $2 \leq n \rightarrow 2 \leq m \rightarrow \exists (e\ e' : edge), Tileable\ n\ m\ b\ e\ e'.$

Proof.

move $\Rightarrow b\ n\ m\ H\ H0.$

$\exists (e_nm\ n\ m\ b).$

$\exists (e'_nm\ n\ m\ b).$

apply $(Pnm_Tileable_nm _ _ _ H\ H0).$

Qed.

Theorem $e_Tileable : \forall (b : boundary)(n m : nat),$
 $2 \leq n \rightarrow 2 \leq m \rightarrow Tileable\ n\ m\ b\ (e_nm\ n\ m\ b)\ (e'_nm\ n\ m\ b).$

Proof.

apply $Pnm_Tileable_nm.$

Qed.

Theorem $Tileable_exists : \forall (b : boundary)(n m : nat),$

$2 \leq n \rightarrow 2 \leq m \rightarrow \exists (e \ e' : \text{edge}), \text{Tileable } n \ m \ b \ e \ e'.$

Proof.

move $\Rightarrow b \ n \ m \ H0 \ H1.$

$\exists (e_nm \ n \ m \ b).$

$\exists (e'_nm \ n \ m \ b).$

apply (e_Tileable b n m H0 H1).

Qed.

1.5 Export to Mathematica

mathematica へのエクスポートのための設定

Definition *null_list* {A : Type} (l m : list A) : Prop.

Proof.

apply True.

Qed.

Notation "{ x }" := (cons x nil).

Notation "{ x , .. , y }" := (cons x .. (cons y nil) ..).

Notation "Tiling[l , m]" := (null_list l m).

Fixpoint *e_list_n* (f : nat → nat)(n : nat) :=

match n with

| 0 \Rightarrow nil

| S i \Rightarrow (e_list_n f i) ++ {f (S i)}

end.

Fixpoint *e_list* (e : edge)(n m : nat) :=

match n with

| 0 \Rightarrow {e_list_n (e 0) m}

| S i \Rightarrow (e_list e i m) ++ {e_list_n (e (S i)) m}

end.

Fixpoint *e'_list_n* (f : nat → nat)(n : nat) :=

match n with

| 0 \Rightarrow {f 0}

| S i \Rightarrow (e'_list_n f i) ++ {f (S i)}

end.

Fixpoint *e'_list* (e : edge)(n m : nat) :=

match n with

| 0 \Rightarrow nil

| S i \Rightarrow (e'_list e i m) ++ {e'_list_n (e (S i)) m}

end.

Definition *tiling_nm2* (n m : nat)(b : boundary) :=

Tiling[e_list (e_nm n m b) n m, e'_list (e'_nm n m b) n m].

Compute (tiling_nm2 4 4 (fun i j \Rightarrow match j with 0 \Rightarrow match i with 2 | 3 \Rightarrow 3 | _ \Rightarrow 4 end

```
| 1 ⇒ 2 | 3 ⇒ 1 | _ ⇒ match i with 0 ⇒ 0 | _ ⇒ 5 end end)).
```

どうしても = と : Prop が邪魔という人向け

```
Ltac print := compute; match goal with ⊢ ?x ⇒ idtac x end.
```

```
Goal (tiling_nm2 4 4 (fun i j ⇒ match j with 0 ⇒ match i with 2 | 3 ⇒ 3 | _ ⇒ 2 end | 1  
⇒ 2 | 3 ⇒ 1 | _ ⇒ match i with 0 ⇒ 0 | _ ⇒ 1 end end)).
```

```
print.
```