

Chapter 1

Library **TilingProgram**

1.1 Preference

```
Require Import Ssreflect.ssreflect Ssreflect.ssrnat Ssreflect.ssrbool Ssreflect.ssrfun Ssreflect.eqtype.  
Require Import ExtrOcamlNatInt ExtrOcamlString.
```

Change the definition of \neq for Prop.

```
Notation "n '!=' m" := ((n == m) = false).
```

```
Lemma eq_or_neq :  $\forall (n\ m : nat), (n == m) \vee (n \neq m)$ .
```

Proof.

```
move  $\Rightarrow$  n m.
```

```
induction (n == m); simpl.
```

```
by [left].
```

```
by [right].
```

Qed.

If you want a color besides $C0$, you can use $C_other2\ C0$.

```
Definition C_other2 (C0 : nat) : nat :=
```

```
match C0 with
```

```
| 0  $\Rightarrow$  1
```

```
| _  $\Rightarrow$  0
```

```
end.
```

```
Lemma C_other2_neq :  $\forall C0 : nat, C0 \neq C\_other2\ C0$ .
```

Proof.

```
case  $\Rightarrow$  [[]]//.
```

Qed.

```
Lemma C_other2_neq' :  $\forall C0 : nat, C\_other2\ C0 \neq C0$ .
```

Proof.

```
case  $\Rightarrow$  [[]]//.
```

Qed.

If you want a color besides $C0$ and $C1$, you can use $C_other3\ C0\ C1$.

Definition $C_other3\ (C0\ C1 : nat) : nat :=$

```
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
```

end.

Lemma $C_other3_neq :$

$\forall (C0\ C1 : nat), C0 \neq C_other3\ C0\ C1 \wedge C1 \neq C_other3\ C0\ C1.$

Proof.

move $\Rightarrow C0\ C1$.

induction $C0$.

induction $C1$.

by [compute].

induction $C1$.

by [compute].

by [compute].

induction $C0$.

induction $C1$.

by [compute].

by [compute].

induction $C1$.

by [compute].

by [compute].

Qed.

Lemma $C_other3_neq1 :$

$\forall (C0\ C1 : nat), C0 \neq C_other3\ C0\ C1.$

Proof.

apply C_other3_neq .

Qed.

Lemma $C_other3_neq2 :$

```

  ∀ (C0 C1 : nat), C1 != C_other3 C0 C1.
Proof.
apply C_other3_neq.
Qed.

Lemma C_other3_neq1' :
  ∀ (C0 C1 : nat), C_other3 C0 C1 != C0.
Proof.
move ⇒ C0 C1.
rewrite eq_sym.
apply C_other3_neq.
Qed.

Lemma C_other3_neq2' :
  ∀ (C0 C1 : nat), C_other3 C0 C1 != C1.
Proof.
move ⇒ C0 C1.
rewrite eq_sym.
apply C_other3_neq.
Qed.

```

1.2 Wang tiling

The boundary and edge functions return a color from x and y coordinates.

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

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

The functions below show the tiling result using edge functions.

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

Proof.

apply x.

Qed.

Notation "''^'" := (null 0).

Notation "''#'" := (null 1).

Open Scope *list_scope*.

Fixpoint *e_i* (j : nat) : edge → nat → list nat :=

fun (e : edge)(i : 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).

```

The functions below returns edge functions e and e' from the size of rectangular region $n \times m$ and boundary function b .

e_12 and e'_12 are tiling functions for P_{12} (1×2 region).

Definition e_12 ($b : boundary$) : $edge$.

The horizontal edges satisfy $e\ 0\ j = b\ 0\ j$ and $e\ 1\ j = b\ 2\ j$.

```

rewrite /edge.
apply (fun i j : nat ⇒
  match i with
  | 0 ⇒ b 0 j
  | _ ⇒ b 2 j
  end).
Defined.

```

Definition e'_12 ($b : boundary$) : $edge$.

The vertical edges satisfy $e'\ 1\ 0 = b\ 1\ 0$ and $e'\ 1\ 2 = b\ 1\ 3$.

```

rewrite /edge.
apply (fun i j : nat ⇒
  match j with
  | 0 ⇒ b 1 0
  | 1 ⇒ if b 1 0 == b 1 3
        then
          (if b 0 1 == b 2 1 then C_other2 (b 1 0) else b 1 0)
        else
          (if b 0 1 == b 2 1
            then
              (if b 0 2 == b 2 2 then C_other3 (b 1 0) (b 1 3) else b 1 3)
            else b 1 0)
  | _ ⇒ b 1 3
  end).
Defined.

```

The followings are tiling functions for P_{22} .

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

```

rewrite /edge.
apply (fun i j : nat =>
match i with
| 0 => b 0 j
| 1 => if b 1 0 == b 1 3
      then
        (if b 2 0 == b 2 3
         then C_other3 (b 0 j) (b 3 j)
         else
          (if b 0 1 == b 3 1
           then b 0 j
           else
            (if b 0 2 == b 3 2
             then (match j with
                  | 0 | 1 => b 3 1
                  | _ => C_other2 (b 0 j)
             end)
            else b 3 j))))
      else
        (if b 2 0 == b 2 3
         then
          (if b 0 1 == b 3 1
           then b 3 j
           else
            (if b 0 2 == b 3 2
             then (match j with
                  | 0 | 1 => b 0 1
                  | _ => C_other2 (b 3 j)
             end)
            else b 0 j))
          else (match j with
               | 0 | 1 => b 0 1
               | _ => b 3 2
             end))
        | _ => b 3 j
end).
Defined.

Definition e'_22 (b : boundary) : edge.
  Define e'_22 using e_22.

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.

```

Divide the $(n+1) \times m$ boundary b to $n \times m$ boundary $\text{bSnm_to_bnm } b$ and $1 \times m$ boundary $\text{bSnm_to_b1m } b$.

Definition $\text{bSnm_to_bnm } (m : \text{nat}) : \text{boundary} \rightarrow \text{boundary}$.

$\text{move} \Rightarrow b$.

$\text{rewrite } / \text{boundary}$.

```

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)

```

end).

Defined.

Definition $\text{bSnm_to_b1m } (m : \text{nat}) : \text{boundary} \rightarrow \text{boundary}$.

$\text{move} \Rightarrow b$.

$\text{rewrite } / \text{boundary}$.

```

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).
Defined.

```

The followings are tiling functions for $1 \times m$ boundary obtained from `bSnm_to_b1m`.

Definition `e_1m` ($b : \text{boundary}$) : `edge`.

The horizontal edges satisfy $e\ 0\ j = b\ 0\ j$ and $e\ 1\ j = b\ 2\ j$.

```

rewrite /edge.
apply (fun i j : nat =>
match i with
| 0 => b 0 j
| _ => b 2 j
end).
Defined.

```

Definition `e'_1m` ($m : \text{nat}$)($b : \text{boundary}$) : `edge`.

We define the colors of vertical edges as $b\ 1\ 0 = e'\ 1\ 0 <> e'\ 1\ 1 <> e'\ 1\ 2 = \dots = e'\ 1\ m$.

```

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).
Defined.

```

The definitions below are tiling functions for P_{n2} .

```

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 ⇒ (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.

```

We can change P_{nm} boundary and edge functions for P_{mn} ones.

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.

The followings are tiling functions for P_{nm} ($|C| ≥ 3 ∧ n, m ≥ 2$).

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 ($n m : nat$) : $boundary → edge$:=


```

fun b : boundary ⇒
match n with
| 0 | 1 ⇒ e'_1m m b
| 2 ⇒ enm_to_emn (fun b' ⇒ e_n2 m b') b
| S n' ⇒ 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' m (bSnm_to_bnm m b) i' j
      end
end.

```

Definition *tiling_nm* ($n m : \text{nat}$)($b : \text{boundary}$) :=
tiling $n m b$ ($e_{nm} n m$) ($e'_{nm} n m$).

1.3 Examples

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 _ _ ⇒ 0) *e_12* *e'_12*).

```

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

```

and ^ express the center and the corner of tiles respectively.

Compute (*tiling* 1 2 (fun _ j ⇒ match j with 1 ⇒ 2 | _ ⇒ 1 end) *e_12* *e'_12*).

```

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

```

Definition *boundary22a* (*i j:nat*) := 0.

Compute (*tiling* 2 2 *boundary22a 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)

```

Definition *boundary22b* (*i j:nat*) :=

match *i* **with** 0 \Rightarrow 2 | _ \Rightarrow **match** *j* **with** 0 \Rightarrow 0 | _ \Rightarrow 1 **end end**.

Compute (*tiling* 2 2 *boundary22b 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)

```

Definition *boundary22c* (*i j:nat*) :=

match *j* **with** 1 \Rightarrow 2 | 3 \Rightarrow 1 | _ \Rightarrow **match** *i* **with** 1 \Rightarrow 0 | _ \Rightarrow 1 **end end**.

Compute (*tiling* 2 2 *boundary22c 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* \Rightarrow *match* *j* with 1 \Rightarrow 2 | 3 \Rightarrow 1 | _ \Rightarrow *match* *i* with 1 \Rightarrow 0 | _ \Rightarrow 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* \Rightarrow *match* *i* with 0 \Rightarrow 2 | _ \Rightarrow *match* *j* with 0 \Rightarrow 0 | _ \Rightarrow 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* \Rightarrow *match* *j* with 1 \Rightarrow 2 | 3 \Rightarrow 1 | _ \Rightarrow *match* *i* with 1 \Rightarrow 0 | _ \Rightarrow 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* \Rightarrow *match* *i* with 0 \Rightarrow 2 | _ \Rightarrow *match* *j* with 0 \Rightarrow 0 | _ \Rightarrow 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)
```

Definition boundary44a (i j:nat) :=

match i with 0 => 2 | 3 => match j with 0 => 5 | _ => 1 end | _ => match j with 1 => 3 | _ => 4 end end.

Compute (tiling_nm 4 4 boundary44a).

```
= (^ :: 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)
```

Definition boundary44b (i j:nat) :=

match j with 0 => match i with 2 | 3 => 3 | _ => 4 end | 1 => 2 | 3 => 1 | _ => match i with 0 => 0 | _ => 5 end end.

Compute (tiling_nm 4 4 boundary44b).

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

Definition *boundary44c* (*i j* : *nat*) :=
 match *j* with 0 ⇒ match *i* with 2 | 3 ⇒ 3 | _ ⇒ 2 end | 1 ⇒ 2 | 3 ⇒ 1 | _ ⇒ match *i*
 with 0 ⇒ 0 | _ ⇒ 1 end end.
Compute (*tiling_nm* 4 4 *boundary44c*).

```

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

```

1.4 Main theorems

The definitions below are the conditions for valid brick Wang tiling.

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

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

These tactics help us to prove the properties of tiling.

Ltac *e_unfold* := **rewrite** /*e_nm*/ *e'_nm*/ *enm_to_emn*/ *e_n2*/ *e'_n2*/ *bnm_to_bmn*/ *e'_22*/ *e_22*/ *e'_12*.

Ltac *eq_rewrite* :=

```

repeat match goal with
| [H : is_true (_ == _) ⊢ _] ⇒ rewrite (elimTF eqP H)
| [H : (_ != _) ⊢ _] ⇒ rewrite H
end;
try repeat rewrite eq_refl.

```

```

Ltac C_other :=
  repeat match goal with
    | [ _ : _ ⊢ _ ] ⇒ rewrite C_other2_neq
    | [ _ : _ ⊢ _ ] ⇒ rewrite C_other2_neq'
    | [ _ : _ ⊢ _ ] ⇒ rewrite C_other3_neq1
    | [ _ : _ ⊢ _ ] ⇒ rewrite C_other3_neq2
    | [ _ : _ ⊢ _ ] ⇒ rewrite C_other3_neq1'
    | [ _ : _ ⊢ _ ] ⇒ rewrite C_other3_neq2'
  end;
  try by [].
Ltac by_or :=
  try repeat (try by [left]; right); by [].
Ltac Brick_auto := intros; e_unfold; eq_rewrite; move ⇒ i j;
  repeat match goal with
    | [ _ : _ ⊢ _ ] ⇒ by_or
    | [ j : nat ⊢ _ ] ⇒ induction j
    | [ i : nat ⊢ _ ] ⇒ induction i
    | [ _ : _ ⊢ _ ] ⇒ progress eq_rewrite
    | [ _ : _ ⊢ _ ] ⇒ progress C_other
  end.

```

Lemmas of the validity of P_{22} tiling.

Lemma *Boundary_i22* : $\forall b : \text{boundary}, \text{Boundary}_i \ 2 \ 2 \ b \ (e_nm \ 2 \ 2 \ b)$.

Proof.

move ⇒ *b*.

case.

by_or.

case.

left.

by [*e_unfold*].

case.

left.

by [*e_unfold*].

by_or.

Qed.

Lemma *Boundary_j22* : $\forall b : \text{boundary}, \text{Boundary}_j \ 2 \ 2 \ b \ (e_nm \ 2 \ 2 \ b)$.

Proof.

move ⇒ *b*.

case.

by_or.

case.

left.

by [*e_unfold*].

case.

left.

by [e_unfold].

by_or.

Qed.

Lemma *Brick22_eexx* : $\forall b : \text{boundary}$,
 $b\ 0\ 1 == b\ 3\ 1 \rightarrow b\ 0\ 2 == b\ 3\ 2 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_enee* : $\forall b : \text{boundary}$,
 $b\ 0\ 1 == b\ 3\ 1 \rightarrow b\ 0\ 2 != b\ 3\ 2 \rightarrow b\ 1\ 0 == b\ 1\ 3 \rightarrow b\ 2\ 0 == b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_enen* : $\forall b : \text{boundary}$,
 $b\ 0\ 1 == b\ 3\ 1 \rightarrow b\ 0\ 2 != b\ 3\ 2 \rightarrow b\ 1\ 0 == b\ 1\ 3 \rightarrow b\ 2\ 0 != b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_enne* : $\forall b : \text{boundary}$,
 $b\ 0\ 1 == b\ 3\ 1 \rightarrow b\ 0\ 2 != b\ 3\ 2 \rightarrow b\ 1\ 0 != b\ 1\ 3 \rightarrow b\ 2\ 0 == b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_ennn* : $\forall b : \text{boundary}$,
 $b\ 0\ 1 == b\ 3\ 1 \rightarrow b\ 0\ 2 != b\ 3\ 2 \rightarrow b\ 1\ 0 != b\ 1\ 3 \rightarrow b\ 2\ 0 != b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_neee* : $\forall b : \text{boundary}$,
 $b\ 0\ 1 != b\ 3\ 1 \rightarrow b\ 0\ 2 == b\ 3\ 2 \rightarrow b\ 1\ 0 == b\ 1\ 3 \rightarrow b\ 2\ 0 == b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_neen* : $\forall b : \text{boundary},$
 $b\ 0\ 1 \neq b\ 3\ 1 \rightarrow b\ 0\ 2 = b\ 3\ 2 \rightarrow b\ 1\ 0 = b\ 1\ 3 \rightarrow b\ 2\ 0 \neq b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_nene* : $\forall b : \text{boundary},$
 $b\ 0\ 1 \neq b\ 3\ 1 \rightarrow b\ 0\ 2 = b\ 3\ 2 \rightarrow b\ 1\ 0 \neq b\ 1\ 3 \rightarrow b\ 2\ 0 = b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_nenn* : $\forall b : \text{boundary},$
 $b\ 0\ 1 \neq b\ 3\ 1 \rightarrow b\ 0\ 2 = b\ 3\ 2 \rightarrow b\ 1\ 0 \neq b\ 1\ 3 \rightarrow b\ 2\ 0 \neq b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_nnee* : $\forall b : \text{boundary},$
 $b\ 0\ 1 \neq b\ 3\ 1 \rightarrow b\ 0\ 2 \neq b\ 3\ 2 \rightarrow b\ 1\ 0 = b\ 1\ 3 \rightarrow b\ 2\ 0 = b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_nnen* : $\forall b : \text{boundary},$
 $b\ 0\ 1 \neq b\ 3\ 1 \rightarrow b\ 0\ 2 \neq b\ 3\ 2 \rightarrow b\ 1\ 0 = b\ 1\ 3 \rightarrow b\ 2\ 0 \neq b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_nnne* : $\forall b : \text{boundary},$
 $b\ 0\ 1 \neq b\ 3\ 1 \rightarrow b\ 0\ 2 \neq b\ 3\ 2 \rightarrow b\ 1\ 0 \neq b\ 1\ 3 \rightarrow b\ 2\ 0 = b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma *Brick22_nnnn* : $\forall b : \text{boundary},$
 $b\ 0\ 1 \neq b\ 3\ 1 \rightarrow b\ 0\ 2 \neq b\ 3\ 2 \rightarrow b\ 1\ 0 \neq b\ 1\ 3 \rightarrow b\ 2\ 0 \neq b\ 2\ 3 \rightarrow$
 $\text{Brick}\ 2\ 2\ (e_nm\ 2\ 2\ b)\ (e'_nm\ 2\ 2\ b).$

Proof.

Brick_auto.

Qed.

Lemma Brick22: $\forall b : \text{boundary}, \text{Brick } 2 \ 2 \ (e_nm \ 2 \ 2 \ b) \ (e'_nm \ 2 \ 2 \ b).$

Proof.

move $\Rightarrow b$.

case ($eq_or_neq \ (b \ 0 \ 1) \ (b \ 3 \ 1)$) $\Rightarrow H$;

case ($eq_or_neq \ (b \ 0 \ 2) \ (b \ 3 \ 2)$) $\Rightarrow H0$;

case ($eq_or_neq \ (b \ 1 \ 0) \ (b \ 1 \ 3)$) $\Rightarrow H1$;

case ($eq_or_neq \ (b \ 2 \ 0) \ (b \ 2 \ 3)$) $\Rightarrow H2$.

apply ($\text{Brick22_eexx } b \ H \ H0$).

apply ($\text{Brick22_eexx } b \ H \ H0$).

apply ($\text{Brick22_eexx } b \ H \ H0$).

apply ($\text{Brick22_eexx } b \ H \ H0$).

apply ($\text{Brick22_enee } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_enen } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_enne } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_ennn } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_neee } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_neen } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_nene } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_nenn } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_nnee } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_nnen } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_nnne } b \ H \ H0 \ H1 \ H2$).

apply ($\text{Brick22_nnnn } b \ H \ H0 \ H1 \ H2$).

Qed.

Lemma P22_Valid_nm : $\forall b : \text{boundary}, \text{Valid_nm } 2 \ 2 \ b.$

Proof.

move $\Rightarrow b$.

repeat split.

apply Boundary_i22 .

apply Boundary_j22 .

apply Brick22 .

Qed.

If P_{2m} ($m \geq 2$) tiling is valid, then $P_{2(m+1)}$ one is also valid.

Lemma replace_e_n2 : $\forall m : \text{nat},$

$2 \leq m \rightarrow (\text{fun } b : \text{boundary} \Rightarrow e_n2 \ (S \ m) \ b) = (\text{fun } b : \text{boundary} \Rightarrow (\text{fun } (i \ j : \text{nat}) \Rightarrow \text{match } i \text{ with } 0 \Rightarrow (bSnm_to_b1m \ 2 \ b) \ 0 \ j \mid S \ i' \Rightarrow e_n2 \ m \ (bSnm_to_bnm \ 2 \ b) \ i' \ j \text{ end})).$

Proof.

move $\Rightarrow m \ H$.

rewrite $/e_n2$.

induction m .

```

discriminate H.
induction m.
discriminate H.
by [].
Qed.

```

Lemma *replace_e'_n2* : $\forall m : \text{nat},$
 $2 \leq m \rightarrow (\text{fun } b : \text{boundary} \Rightarrow e'_n2 (S\ m)\ b) = (\text{fun } b : \text{boundary} \Rightarrow (\text{fun } (i\ j : \text{nat}) \Rightarrow$
 $\text{match } i \text{ with } 0 \Rightarrow 0 \mid 1 \Rightarrow e'_{-1}m\ 2\ (bSnm_to_b1m\ 2\ b)\ 1\ j \mid S\ i' \Rightarrow e'_n2\ m\ (bSnm_to_bnm$
 $2\ b)\ i'\ j\ \text{end})).$

Proof.
move $\Rightarrow m\ H$.
rewrite $/e'_n2$.
induction *m*.
discriminate *H*.
induction *m*.
discriminate *H*.
by [].
Qed.

Lemma *Boundary_i_ind_2m* :
 $\forall (b : \text{boundary})(m : \text{nat}), 2 \leq m \rightarrow$
 $(\forall b' : \text{boundary}, \text{Boundary}_i\ 2\ m\ b' (e'_{-nm}\ 2\ m\ b')) \rightarrow$
 $\text{Boundary}_i\ 2\ (S\ m)\ b (e'_{-nm}\ 2\ (S\ m)\ b).$

Proof.
move $\Rightarrow b\ m\ H\ H0$.
rewrite $/e'_{-nm}/e_{-nm}$.
rewrite $(\text{replace}_e_n2\ m\ H)$.
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 $/\text{Boundary}_i/e'_{-nm}/e_{-nm}/enm_to_emn/bnm_to_bmn/bSnm_to_b1m/bSnm_to_bnm$.
move $\Rightarrow H1\ i$.
induction *i*.
by_or.
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-or.
Qed.

```

Lemma *Boundary-j-ind-2m* :

$$\forall (b : \text{boundary})(m : \text{nat}), 2 \leq m \rightarrow$$

$$(\forall b' : \text{boundary}, \text{Boundary_j } 2\ m\ b' (e_nm\ 2\ m\ b')) \rightarrow$$

$$\text{Boundary_j } 2\ (S\ m)\ b\ (e_nm\ 2\ (S\ m)\ b).$$

Proof.

```

move  $\Rightarrow b\ m\ H\ H0$ .
rewrite / $e\_nm$ /.
rewrite ( $replace\_e\_n2\ m\ H$ ).
induction  $m$ .
discriminate  $H$ .
induction  $m$ .
discriminate  $H$ .
clear  $IHm\ IHm0$ .
move : ( $H0\ (b_{nm\_to\_bmn}\ (b_{Snm\_to\_bnm}\ 2\ (b_{nm\_to\_bmn}\ b)))$ ).
rewrite / $Boundary\_j$  /  $e\_nm$  /  $e\_nm$  /  $enm\_to\_emn$  /  $bnm\_to\_bmn$  /  $b_{Snm\_to\_b1m}$  /  $b_{Snm\_to\_bnm}$  /.
move  $\Rightarrow H1\ j$ .
induction  $j$ .
by-or.
induction  $j$ .
rewrite / $e\_1m$ /.
by [left].
case ( $H1\ (S\ j)$ )  $\Rightarrow H2$ .
left.
apply  $H2$ .
case  $H2 \Rightarrow H3$ .
discriminate  $H3$ .
repeat right.
apply  $H3$ .
Qed.

```

Lemma *Brick-ind-2m* :

$$\forall (b : \text{boundary})(m : \text{nat}), 2 \leq m \rightarrow$$

$$(\forall b' : \text{boundary}, \text{Brick } 2\ m\ (e_nm\ 2\ m\ b') (e_nm\ 2\ m\ b')) \rightarrow$$

Brick 2 (S m) (e_nm 2 (S m) b) (e'_nm 2 (S m) b).

Proof.

```

move ⇒ b m H H0.
rewrite /e'_nm/e_nm.
rewrite (replace_e_n2 m H).
rewrite (replace_e'_n2 m H).
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 /Brick/e'_nm/e_nm/enm_to_emn/bnm_to_bmn/bSnm_to_b1m/bSnm_to_bnm.
move ⇒ H1 i j.
induction m.
induction j.
induction i.
rewrite /e_n2/e_22/e'_1m.
C_other.
by_or.
induction i.
rewrite /e_n2/e_22/e'_1m.
C_other.
by_or.
by_or.
apply H1.
induction j.
induction i.
rewrite /e_n2/e_22/e'_1m/bSnm_to_b1m.
C_other.
by_or.
induction i.
rewrite /e_n2/e_22/e'_1m/bSnm_to_b1m.
C_other.
by_or.
by_or.
apply H1.

```

Qed.

Lemma *Valid_nm_ind_2m* : $\forall (b : \text{boundary})(m : \text{nat}),$
 $2 \leq m \rightarrow (\forall b' : \text{boundary}, \text{Valid_nm } 2 \ m \ b') \rightarrow \text{Valid_nm } 2 \ (S \ m) \ b.$

Proof.

move ⇒ b m H H0.

```

split.
apply (Boundary_i_ind_2m _ _ H).
apply H0.
split.
apply (Boundary_j_ind_2m _ _ H).
apply H0.
apply (Brick_ind_2m _ _ H).
apply H0.
Qed.

```

If $m \geq 2$, then P_{2m} tiling is always valid.

Lemma $P_{2m_Valid_nm} : \forall (b : boundary)(m : nat), 2 \leq m \rightarrow Valid_nm \ 2 \ m \ b$.

Proof.

```

induction m.
discriminate.
induction m.
discriminate.
clear IHm IHm0.
move : b.
induction m.
move => b H.
apply P22_Valid_nm.
move => b H.
apply Valid_nm_ind_2m.
apply H.
move => b'.
apply IHm.
apply H.
Qed.

```

If P_{nm} ($n, m \geq 2$) tiling is valid, then $P_{(n+1)m}$ one is also valid.

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

$e_nm \ n.+3 \ m \ b = (\text{fun } (i \ j : nat) \Rightarrow \text{match } i \text{ with } 0 \Rightarrow (bSnm_to_b1m \ m \ b) \ 0 \ j \mid S \ i' \Rightarrow e_nm \ n.+2 \ m \ (bSnm_to_bnm \ m \ b) \ i' \ j \text{ end}).$

Proof.

```

move => b n m.
by [rewrite /e_nm].
Qed.

```

Lemma $replace_e'_nm : \forall (b : boundary)(n \ m : nat),$

$e'_nm \ n.+3 \ m \ b = (\text{fun } (i \ j : nat) \Rightarrow \text{match } i \text{ with } 0 \Rightarrow 0 \mid 1 \Rightarrow e'_1m \ m \ (bSnm_to_b1m \ m \ b) \ 1 \ j \mid S \ i' \Rightarrow e'_nm \ n.+2 \ m \ (bSnm_to_bnm \ m \ b) \ i' \ j \text{ end}).$

Proof.

```

move => b n m.

```

by [rewrite /e'_nm].

Qed.

Lemma *Boundary-i-ind-nm* :

$\forall (b : \text{boundary})(n\ m : \text{nat}), 2 \leq n \rightarrow 2 \leq m \rightarrow$
 $(\forall b' : \text{boundary}, \text{Boundary-}i\ n\ m\ b' (e'_{-nm}\ n\ m\ b')) \rightarrow$
 $\text{Boundary-}i\ (S\ n)\ m\ b (e'_{-nm}\ (S\ n)\ m\ b).$

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 /*Boundary-i*.

move \Rightarrow *H2*.

induction *i*.

by_or.

move : (*H2 i*).

rewrite *replace-e'_nm*.

induction *m*.

discriminate *H0*.

induction *m*.

discriminate *H0*.

clear *IHm IHm0*.

case \Rightarrow *H3*.

left.

induction *i*.

by [rewrite /e'_1m/bSnm_to_b1m/bSnm_to_bnm].

move : *H3*.

by [rewrite /bSnm_to_bnm].

case *H3* \Rightarrow *H4*.

rewrite *H4*.

rewrite /e'_1m/bSnm_to_b1m/bSnm_to_bnm.

by_or.

repeat right.

apply *H4*.

Qed.

Lemma *Boundary-j-ind-nm* :

$\forall (b : \text{boundary})(n\ m : \text{nat}), 2 \leq n \rightarrow 2 \leq m \rightarrow$
 $(\forall b' : \text{boundary}, \text{Boundary-}j\ n\ m\ b' (e_{-nm}\ n\ m\ b')) \rightarrow$
 $\text{Boundary-}j\ (S\ n)\ m\ b (e_{-nm}\ (S\ n)\ m\ b).$

Proof.

```
move ⇒ 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 /Boundary_j.
move ⇒ H2 j.
move : (H2 j).
induction j.
by_or.
rewrite replace_e_nm.
induction m.
discriminate H0.
induction m.
discriminate H0.
clear IHm IHm0.
case ⇒ H4.
left.
split.
by [rewrite /bSnm_to_bnm/bSnm_to_b1m].
rewrite /bSnm_to_bnm/bSnm_to_b1m in H4.
apply H4.
right.
apply H4.
Qed.
```

Lemma *Brick_ind_nm* :

$$\begin{aligned} & \forall (b : \text{boundary})(n\ m : \text{nat}), 2 \leq n \rightarrow 2 \leq m \rightarrow \\ & (\forall b' : \text{boundary}, \text{Valid_nm } n\ m\ b') \rightarrow \\ & \text{Brick } (S\ n)\ m\ (e_nm\ (S\ n)\ m\ b)\ (e'_nm\ (S\ n)\ m\ b). \end{aligned}$$

Proof.

```
move ⇒ 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 /Valid_nm/Boundary_i/Boundary_j/Brick.
elim ⇒ H2.
```

```

elim  $\Rightarrow$   $H3$   $H4$ .
clear  $H2$ .
rewrite  $replace\_e\_nm$   $replace\_e'\_nm$ .
move  $\Rightarrow$   $i$   $j$ .
induction  $i$ .
case ( $H3$   $j$ .+1)  $\Rightarrow$   $H5$ .
elim  $H5$   $\Rightarrow$   $H6$   $H7$ .
rewrite ( $elimTF$   $eqP$   $H6$ ).
induction  $m$ .
discriminate  $H0$ .
induction  $m$ .
discriminate  $H0$ .
clear  $IHm$   $IHm0$ .
rewrite  $/bSnm\_to\_b1m/$   $bSnm\_to\_bnm/$   $enm\_to\_emn/$   $bnm\_to\_bmn/$   $e'\_1m$ .
induction  $j$ .
 $C\_other$ .
 $by\_or$ .
induction  $j$ .
 $C\_other$ .
 $by\_or$ .
 $C\_other$ .
 $by\_or$ .
case  $H5$   $\Rightarrow$   $H6$ .
discriminate  $H6$ .
repeat right.
apply  $H6$ .
apply  $H4$ .
Qed.

Lemma  $Valid\_nm\_ind\_nm : \forall (b : boundary)(n\ m : nat),$ 
 $2 \leq n \rightarrow 2 \leq m \rightarrow (\forall b' : boundary, Valid\_nm\ n\ m\ b') \rightarrow Valid\_nm\ (S\ n)\ m\ b$ .
Proof.
move  $\Rightarrow$   $b\ n\ m\ H\ H0\ H1$ .
split.
apply ( $Boundary\_i\_ind\_nm$   $--$   $H\ H0$ ).
apply  $H1$ .
split.
apply ( $Boundary\_j\_ind\_nm$   $--$   $H\ H0$ ).
apply  $H1$ .
apply ( $Brick\_ind\_nm$   $--$   $H\ H0$ ).
apply  $H1$ .
Qed.

```

If $n, m \geq 2$, then P_{nm} tiling is always valid.

Theorem *e_nm_Valid* : $\forall (b : \text{boundary})(n\ m : \text{nat}),$
 $2 \leq n \rightarrow 2 \leq m \rightarrow \text{Valid}\ n\ m\ b\ (e_nm\ n\ m\ b)\ (e'_nm\ n\ m\ b).$

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_Valid_nm _ _ H0).
move  $\Rightarrow$  b.
apply Valid_nm_ind_nm.
apply H.
apply H0.
apply IHn.
apply H.
```

Qed.

Theorem *P22_Tileable* : $\forall (b : \text{boundary})(n\ m : \text{nat}),$
 $2 \leq n \rightarrow 2 \leq m \rightarrow \exists (e\ e' : \text{edge}), \text{Valid}\ n\ m\ b\ e\ e'.$

Proof.

```
move  $\Rightarrow$  b n m H0 H1.
 $\exists$  (e_nm n m b).
 $\exists$  (e'_nm n m b).
apply (e_nm_Valid b n m H0 H1).
```

Qed.

1.5 Export to Mathematica

You can export the tiling results to Mathematica (Use /old_files/Tiling.nb).

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 ⇒ nil
  | S i ⇒ (e_list_n f i) ++ {f (S i)}
  end.
Fixpoint e_list (e : edge)(n m : nat) :=
  match n with
  | 0 ⇒ {e_list_n (e 0) m}
  | S i ⇒ (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 ⇒ {f 0}
  | S i ⇒ (e'_list_n f i) ++ {f (S i)}
  end.
Fixpoint e'_list (e : edge)(n m : nat) :=
  match n with
  | 0 ⇒ nil
  | S i ⇒ (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 ⇒ match j with 0 ⇒ match i with 2 | 3 ⇒ 3 | _ ⇒ 4 end
| 1 ⇒ 2 | 3 ⇒ 1 | _ ⇒ match i with 0 ⇒ 0 | _ ⇒ 5 end end)).

You can remove = and : Prop.

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.
Abort.

Extraction "TilingProgram.ml" tiling_nm boundary22a boundary22b boundary22c bound-
ary44a boundary44b boundary44c.

```