Chapter 1

Library TilingProgram2

1.1 Preliminary (1)

```
Require Import Ssreflect.ssreflect Ssreflect.eqtype Ssreflect.ssrbool Ssreflect.ssrnat Ssreflect.ssrfun.
Coercion istrue (b : bool) := is_true b.

Lemma neq_nm: ∀ (n m : nat), (n ≠ m) → ((n < m) ∨ (m < n)).
Proof.
move ⇒ n m H.
move: (leq_total n m).
move/orP ⇒ H1.
case H1.
rewrite (leq_eqVlt n m).
move/orP.
elim.
move/eqP ⇒ H2.
by move: (H H2).
move ⇒ H2.
by left.
rewrite (leq_eqVlt m n).
move/orP.
elim.
move/eqP ⇒ H2.
rewrite H2 in H.
by move: (H (erefl n)).
move ⇒ H2.
by right.
Qed.

Lemma neq_S: ∀ m:nat, m ≠ m.+1.
Proof.
```

```
elim.
by [].
move => n H.
case.
apply H.
Qed.
```

```
Lemma neq_SS: ∀ (n m:nat), n.+1 ≠ m.+1 ↔ n ≠ m.
Proof.
move => n m.
split.
move: n m.
elim.
elim.
by [].
elim.
by [].
move => n H H1 H2.
by [].
move => n H m H1.
move/eqP in H1.
rewrite eqSS in H1.
move/eqP in H1.
apply H1.
move => H.
move/eqP.
rewrite eqSS.
move/eqP.
apply H.
Qed.
```

```
Lemma eq_or_neq: ∀ (n m:nat), n = m ∨ n ≠ m.
Proof.
elim.
elim.
by left.
by right.
move => n H.
case.
by right.
move => n0.
move: (H n0) => H1.
case H1.
move => H2.
```

```
left.
by rewrite H2.
move => H2.
right.
apply (neq_SS n n0).
apply H2.
Qed.

Lemma x_eq_y_and_x_to_y (a b: Prop):
  (a <-> b) -> a -> b.
Proof.
move => H a0.
apply H.
apply a0.
Qed.

Lemma x_eq_y_and_y_to_x (a b: Prop):
  (a <-> b) -> b -> a.
Proof.
move => H b0.
apply H.
apply b0.
Qed.

Lemma x_eq_y_and_not_x_to_not_y (a b:Prop):
  (a <-> b) -> (~ a) -> (~ b).
Proof.
move => H a0 b0.
apply (a0 (@x_eq_y_and_y_to_x a b H b0)).
Qed.

Lemma x_eq_y_and_not_y_to_not_x (a b:Prop):
  (a <-> b) -> (~ b) -> (~ a).
Proof.
move => H b0 a0.
apply (b0 (@x_eq_y_and_x_to_y _ _ H a0)).
Qed.
```

1.2 Preliminary (2)

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_eq : ∀ n, 0 = eq_to_bin n n.
Proof.
elim ⇒ [|n H] //.
Qed.

Lemma eq_to_bin_nn : ∀ n, eq_to_bin n n = 0.
Proof.
move ⇒ n.
by rewrite -(eq_to_bin_eq n).
Qed.

Lemma eq_to_bin_ex : ∀ n m, eq_to_bin n m = eq_to_bin m n.
Proof.
elim.
elim ⇒ [|n H] //.
move ⇒ n H.
elim ⇒ [//|n0 H1] //.
simpl.
apply (H n0).
Qed.

Lemma eq_to_bin_iff1 : ∀ (n m : nat), m < n → 1 = eq_to_bin n m.
Proof.
elim.
elim ⇒ [|] //.
move ⇒ n H.
elim ⇒ [|n0 H1 H2] //.
simpl.
apply (H n0 H2).
Qed.

Lemma eq_to_bin_iff2: ∀ (n m : nat), n < m → 1 = eq_to_bin n m.
Proof.
move ⇒ n m H.
rewrite eq_to_bin_ex.
apply (eq_to_bin_iff1 m n H).
Qed.

Lemma eq_to_bin_iff3: ∀ (n m : nat), (n ≠ m) → 1 = eq_to_bin n m.
Proof.
 move ⇒ n m H.
move: (neq_nm n m H).
elim ⇒ [/eq_to_bin_iff2|/eq_to_bin_iff1] //.
```

```
Qed.
Lemma eq_to_bin_iff4: ∀ (n m : nat), 1 = eq_to_bin n m → n ≠ m.
Proof.
move ⇒ n m H H1.
rewrite H1 in H.
rewrite -(eq_to_bin_eq m) in H.
discriminate.
Qed.
Lemma eq_to_bin_iff5: ∀ (n m : nat), 0 = eq_to_bin n m → n = m.
Proof.
elim.
elim ⇒ [|n H] //.
move ⇒ n H.
elim⇒ [| n0 H1] //.
simpl.
move ⇒ H2.
move: (H n0 H2) ⇒ H3.
by rewrite H3.
Qed.
Lemma eq_to_bin_iff: ∀ (n m : nat), n = m ↔ 0 = eq_to_bin n m.
Proof.
move ⇒ n m.
split.
move ⇒ H.
rewrite H.
apply eq_to_bin_eq.
apply eq_to_bin_iff5.
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 : nat) : nat :=
match eq_to_bin n m with
  | 0 ⇒ p
  | _ ⇒ q
end.
```

特定の $C0$ 以外なら何でもいい場合は, `C_other2 C0` で他の色を求める.

```
Definition C_other2 (C0 : nat) : nat :=
match C0 with
  | 0 ⇒ 1
  | _ ⇒ 0
end.
```

```
Lemma C_other2_neq : ∀ C0 : nat, C0 ≠ C_other2 C0.
Proof.
elim ⇒ [|] //.
Qed.
```

```
Lemma C_other2_neq' : ∀ C0 : nat, C_other2 C0 ≠ C0.
Proof.
elim ⇒ [|] //.
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 : 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.
```

```
Lemma C_other3_neq :
 ∀ (C0 C1 : nat), C0 ≠ C_other3 C0 C1 ∧ C1 ≠ C_other3 C0 C1.
Proof.
elim.
elim ⇒ [|] //.
elim ⇒ [|] //.
elim ⇒ [H [|] | n H1 H2 [|]]//.
Qed.
```

```
Lemma C_other3_neq' :
 ∀ (C0 C1 : nat), C_other3 C0 C1 ≠ C0 ∧ C_other3 C0 C1 ≠ C1.
Proof.
elim.
elim ⇒ [|] //.
elim ⇒ [|] //.
elim ⇒ [H [|] | n H1 H2 [|]] //.
Qed.
```

1.3 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 color := nat.
Definition vertical_index := nat.
Definition horizontal_index := nat.
Definition boundary := vertical_index → horizontal_index → color.
Definition edge := vertical_index → horizontal_index → color.
```

```
Definition Boundary_vertical (n m : nat) (b : boundary) (e' : edge) :=
 ∀ i : vertical_index, e' i 0 = b i 0 ∧ e' i m = b i (S m) ∨ i = 0 ∨ n < i.
Definition Boundary_horizontal (n m : nat) (b : boundary) (e : edge) :=
 ∀ j : horizontal_index, 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) :=
 ∀ i : vertical_index, ∀ j : horizontal_index,
 (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 ValidTiling (n m : nat)(b : boundary)(e e' : edge) :=
 Boundary_vertical n m b e' ∧ Boundary_horizontal n m b e ∧ Brick n m e e'.
```

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.
```

```
         e_12 0 1   e_12 0 2
         +- b 0 1 -+-- b 0 2 --+
    b 1 0|         |           | b 1 3
e'_12 1 0|         | e'_12 1 1 | e'_12 1 2
         |         |           |
         +- b 2 1 -+---b 2 2 --+
         e_12 1 1   e_12 1 2

(e_12  0 1) = (b 0 1) (e_12  0 2) = (b 0 2)
(e_12  1 1) = (b 2 1) (e_12  1 2) = (b 2 2)
(e'_12 1 0) = (b 1 0)
(e'_12 1 2) = (b 1 3)
(e'_12 1 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)
```

```
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.
```

```
Lemma eq_to_if_1: ∀ (n p q: nat), (eq_to_if n n p q) = p.
Proof.
case.
by compute.
move ⇒ n p q.
rewrite /eq_to_if.
by rewrite (eq_to_bin_nn n.+1).
Qed.
```

```
Lemma eq_to_if_2: ∀ (n m p q: nat), (n ≠ m) → (eq_to_if n m p q) = q.
Proof.
move ⇒ n m p q H.
rewrite /eq_to_if.
by rewrite -(eq_to_bin_iff3 n m H).
Qed.
```

```
Ltac eq_simpl :=
 repeat match goal with
          | [ _ : _ ⊢ _ ] ⇒ rewrite eq_to_bin_nn
          | [ H : _ ≠ _ ⊢ _ ] ⇒ rewrite -(eq_to_bin_iff3 _ _ H)
```

```
end.
```

1.4 Lemmas and Theorems

```
Lemma e12_Tileable_horizontal : ∀ (b : boundary), Boundary_horizontal 1 2 b (e_12 b).
Proof.
move ⇒ b.
rewrite /Boundary_horizontal/e_12.
case.
by right;left.
case.
by left.
move ⇒ n.
by left.
Qed.
```

```
Lemma e12_Tileable_vertical : ∀ (b : boundary), Boundary_vertical 1 2 b (e'_12 b).
Proof.
move ⇒ b.
rewrite /Boundary_vertical/e'_12.
case.
by right;left.
case.
by left.
move ⇒ n.
by right;right.
Qed.
```

```
Lemma e12_Tileable_brick1 : ∀ (b : boundary),
 ((b 1 0) = (b 1 3)) → (((b 0 1) = (b 2 1)) ↔ ((b 0 2) = (b 2 2))) → (Brick 1 2 (e_12 b)
(e'_12 b)).
Proof.
move ⇒ b H1 H2.
rewrite /Brick.
case.
case.
rewrite /e_12/e'_12.
move: (eq_or_neq (b 1 0) (b 1 3)) ⇒ H3.
case H3.
move ⇒ H4.
rewrite H4.
rewrite /eq_to_if.
eq_simpl.
```

```
move: (eq_or_neq (b 0 1) (b 2 1)) => H5.
case H5.
move => H6.
rewrite -H6.
eq_simpl.
left.
split.
by [].
rewrite /C_other2.
elim (b 1 3) => [|] //.
move => H6.
eq_simpl.
right;left.
split.
apply H6.
by [].
move => H4.
rewrite /eq_to_if.
eq_simpl.
move: (eq_or_neq (b 0 1) (b 2 1)) => H5.
case H5.
move => H6.
rewrite H6.
eq_simpl.
move: (x_eq_y_and_x_to_y _ _ H2 H6) => H7.
rewrite H7.
eq_simpl.
left.
split => [|] //.
move => H6.
eq_simpl.
right;left.
apply (conj H6 (erefl (b 1 0))).
case.
rewrite /e_12/e'_12.
rewrite /eq_to_if.
rewrite H1.
eq_simpl.
move: (eq_or_neq (b 0 1) (b 2 1)) => H3.
case H3.
move => H4.
rewrite H4.
```

```
eq_simpl.
move: (x_eq_y_and_x_to_y _ _ H2 H4) => H5.
left.
split => [|/C_other2_neq'] //.
move => H4.
eq_simpl.
right;left.
split => [/(x_eq_y_and_not_x_to_not_y _ _ H2 H4)|] //.
by right;right;right.
by right;right;left.
Qed.

Lemma e12_Tileable_brick2 : ∀ (b : boundary),
 ((b 1 0) ≠ (b 1 3)) → (((b 0 1) = (b 2 1)) ∨ ((b 0 2) = (b 2 2)))
  → (Brick 1 2 (e_12 b) (e'_12 b)).
Proof.
move => b H H1.
rewrite /Brick.
case.
case.
rewrite /e_12/e'_12.
rewrite /eq_to_if.
rewrite -(eq_to_bin_iff3 (b 1 0) (b 1 3) H).
move: (eq_or_neq (b 0 1) (b 2 1)) => H2.
case H2.
  move => H3.
  left.
  rewrite H3.
  rewrite (eq_to_bin_nn (b 2 1)).
  split.
  by [].
  case (eq_to_bin (b 0 2) (b 2 2)).
  apply (proj1 (C_other3_neq (b 1 0) (b 1 3))).
  move => n0.
  apply H.

  move => H3.
  right.
  rewrite -(eq_to_bin_iff3 (b 0 1) (b 2 1) H3).
  left.
  split => [/H3|] //.

case.
rewrite /e_12/e'_12.
rewrite /eq_to_if.
```

```
move: (eq_or_neq (b 0 2) (b 2 2)) ⇒ H2.
case H2.
  move ⇒ H3.
  left.
  rewrite -(eq_to_bin_iff3 (b 1 0) (b 1 3) H).
  rewrite H3.
  rewrite (eq_to_bin_nn (b 2 2)).
  split.
  by [].
  case (eq_to_bin (b 0 1) (b 2 1)).
  apply (proj2 (C_other3_neq' (b 1 0) (b 1 3))).
  move ⇒ n.
  apply H.

  move ⇒ H3.
  right.
  rewrite -(eq_to_bin_iff3 (b 1 0) (b 1 3) H).
  rewrite -(eq_to_bin_iff3 (b 0 2) (b 2 2) H3).
  left.
  split.
  by [].
  case H1.
    move ⇒ H4.
    rewrite H4.
    rewrite (eq_to_bin_nn (b 2 1)).
    by [].

    move ⇒ H4.
    move: (H3 H4).
    by [].

by right;right;right.
by right;right;left.
Qed.

Theorem e12_ValidTiling_1 :
  ∀ (b : boundary),
 ((b 1 0) = (b 1 3)) → (((b 0 1) = (b 2 1)) ↔ ((b 0 2) = (b 2 2)))
  → ∃ (e e' : edge), ValidTiling 1 2 b e e'.
Proof.
move ⇒ b.
move ⇒ H1 H2.
∃ (e_12 b).
∃ (e'_12 b).
split.
```

```
apply e12_Tileable_vertical.
split.
apply e12_Tileable_horizontal.
by apply e12_Tileable_brick1.
Qed.

Theorem e12_ValidTiling_2 :
  ∀ (b : boundary),
 ((b 1 0) ≠ (b 1 3)) → (((b 0 1) = (b 2 1)) ∨ ((b 0 2) = (b 2 2)))
  → ∃ (e e' : edge), ValidTiling 1 2 b e e'.
Proof.
move ⇒ b.
move ⇒ H1 H2.
∃ (e_12 b).
∃ (e'_12 b).
split.
apply e12_Tileable_vertical.
split.
apply e12_Tileable_horizontal.
by apply e12_Tileable_brick2.
Qed.
```

```
             [2]         [1]
         +- b 0 1 -+-- b 0 2 --+
    b 1 0|         |           | b 1 3
     [1] |         |           |   [1]
         +- b 2 1 -+---b 2 2 --+
             [1]         [1]
```

```
Function counter_example_12 (i j:nat)
  :=
     match i with
       | 0 ⇒ match j with
                | 0 ⇒ 2
                | 1 ⇒ 1
                | 2 ⇒ 0
                | _ ⇒ 2
              end
       | 1 ⇒ match j with
                | 0 ⇒ 0
                | 1 ⇒ 2
                | 2 ⇒ 2
                | 3 ⇒ 0
                | _ ⇒ 2
```

```
            end
      | 2 ⇒ match j with
              | 0 ⇒ 2
              | 1 ⇒ 0
              | 2 ⇒ 0
              | _ ⇒ 2
            end
      | _ ⇒ 2
    end.

Theorem e12_not_ValidTiling :
  ∃ (b : boundary), ∀ (e e':edge), ¬ ValidTiling 1 2 b e e'.
Proof.
  ∃ counter_example_12.
Admitted.

Theorem e22_ValidTiling:
  ∀ (b : boundary), ∃ (e e' : edge), ValidTiling 2 2 b e e'.
Proof.
Admitted.
```