TPPmark 2025

Author: Makoto Kanazawa, Hosei University

Last updated: 2025/10/30

This is an Agda formalization of TPPmark 2025. Tested with Agda 2.8.0 and the Agda standard library v2.3.

```
{-# OPTIONS --rewriting #-}

open import Agda.Builtin.Equality
open import Agda.Builtin.Equality.Rewrite
open import Data.Bool using (Bool; true; false; not; _∧_; _xor_; if_then_else_)
open import Data.Bool.Properties
  using (not-involutive; not-distribˡ-xor; not-distribʳ-xor)
open import Data.Empty using (⊥-elim)
open import Data.Fin using (Fin) renaming (suc to 1+)
open import Data.Fin.Patterns
open import Data.Fin.Properties using (_≟_)
open import Data.List
  using (List; []; _∷_; [_]; _++_; concat; concatMap; map; foldr; cartesianProduct)
open import Data.List.Membership.Propositional using (_∈_; _∉_)
open import Data.List.Membership.Propositional.Properties
  using (∈-++⁻; ∈-map⁻; ∈-cartesianProductWith⁻)
open import Data.List.Properties
  using (++-assoc; cartesianProductWith-distribʳ-++; map-++; foldr-++ ; concat-++)
open import Data.List.Relation.Unary.Any using (here; there)
open import Data.Nat using (ℕ; zero; suc)
open import Data.Product using (_×_; _,_; proj₁; proj₂; ∃; ∃₂)
open import Data.Sum using (_⊎_; inj₁; inj₂)
open import Function using (id; _∘_; flip)
open import Relation.Binary.PropositionalEquality
  using (_≡_; _≢_; _≗_; cong; cong₂; cong-app; sym; trans; module ≡-Reasoning)
open import Relation.Nullary using (yes; no; does)
```

Here's a verbatim quote of the problem:

> Let $n \geq 1$ be an integer. Arrange n^3 identical cube-shaped lamps tightly, forming a large $n \times n \times n$ cube. On each outer face, an $n \times n$ grid of small squares (the exposed faces of the lamps on that face) is visible, for a total of $6n^2$ squares. Each lamp is either on or off. When one of the small squares on the outer surface is pressed, the n lamps on the straight line joining the pressed square and the square directly opposite it on the far face (*1) toggle simultaneously (on↔off).
>
> Given the on/off configuration of all lamps, describe as concisely as possible a necessary and sufficient condition on the configuration to be turned off by repeated applications of the above operation, and prove the correctness of the condition.
>
> (*1) Clarification (Sep 16, 2025): "on the straight line joining the pressed square and the square directly opposite it on the far face" means "on a straight line which vertically penetrates the pressed square".
>
> Credit: Keisuke Nakano

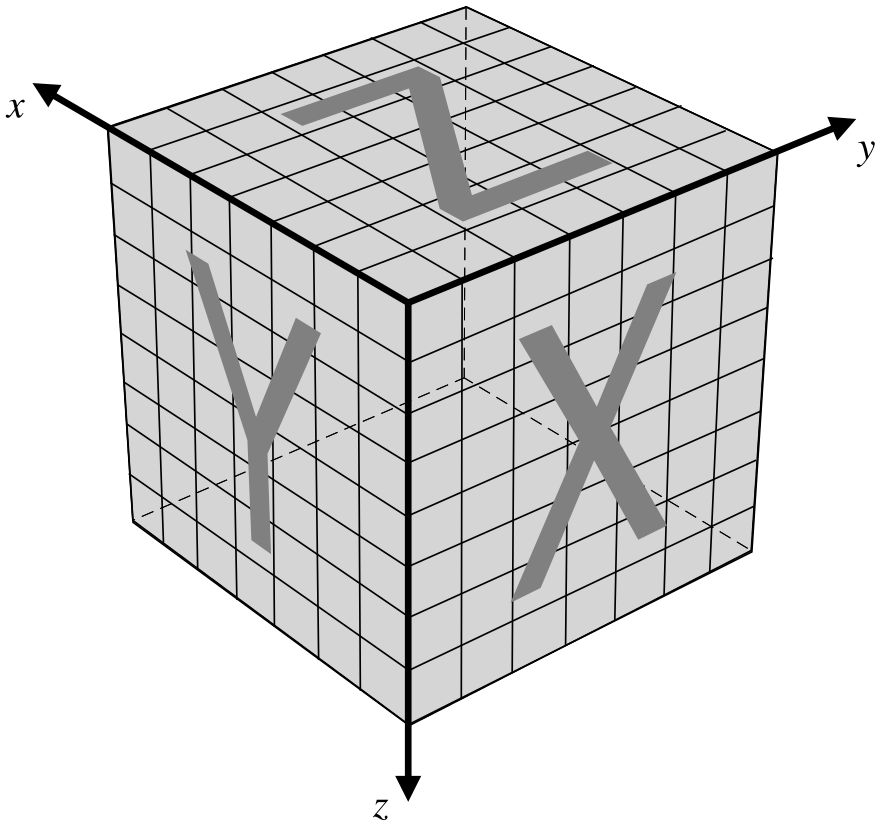
```
module TPPmark (n₁ : ℕ) where
```

```
n : ℕ
n = suc n₁

_≡ᵇ_ : Fin n → Fin n → Bool
i ≡ᵇ j = does (i ≟ j)

≟-refl : (i : Fin n) → (i ≟ i) ≡ yes refl
≟-refl i with i ≟ i
... | no ¬p = ⊥-elim (¬p refl)
... | yes refl = refl

xor-not-rewrite¹ : (x y : Bool) → not x xor y ≡ not (x xor y)
xor-not-rewrite¹ x y = sym (not-distrib¹-xor x y)

xor-not-rewriteʳ : (x y : Bool) → x xor (not y) ≡ not (x xor y)
xor-not-rewriteʳ x y = sym (not-distribʳ-xor x y)

not-not-rewrite : (x : Bool) → not (not x) ≡ x
not-not-rewrite = not-involutive

{-# REWRITE ≟-refl xor-not-rewrite¹ xor-not-rewriteʳ not-not-rewrite #-}
```

In Agda, `Fin n` is the type of natural numbers less than `n`. The elements of `Fin n` are written `0F`, `1F`, `2F`, etc. We represent each of the n^3 cube-shaped lamps by a triple `(i , j , k)`, where `i j k : Fin n`. A *configuration* is a function from `Fin n × Fin n × Fin n` to `Bool`, where `false` means *off* and `true` means *on*. The desired target configuration is one in which all lamps are off.

```
Lamp : Set
Lamp = Fin n × Fin n × Fin n

Configuration : Set
Configuration = Lamp → Bool

AllOff : Configuration → Set
AllOff c = (i j k : Fin n) → c (i , j , k) ≡ false
```

Next we need to decide how to represent the small squares on the outer faces of the large $n \times n \times n$ cube. Let us assume that one of the vertices of the lamp `(0F , 0F , 0F)` lies at the origin of the xyz coordinate system on the three-dimensional space, and the vertex of the lamp `(i , j , k)` that is closest to the origin is the point (i,j,k). We refer to the outer face of the large cube that is on the plane $x = 0$ as `X`, and denote each square on that face by a triple of the form `(X , j , k)`. This triple is (one of) the exposed face(s) of the lamp `(0F , j , k)`. (When either `j` or `k` is `0F`, this lamp has more than one exposed face.) Likewise, the face of the large cube that is on the plane $y = 0$ is `Y`, and the exposed face of the lamp `(i , 0F , k)` on the face `Y` is denoted by `(Y , i , k)`, and similarly for `Z` and `(Z , i , j)`.



We always adopt the vantage point of the above picture and call a lamp *visible* just in case at least one of its coordinates is `0F`. The light gray color of each visible lamp in the picture is meant to indicate that they are either on or off. We do not bother to name the squares on the three faces that are opposite to `X`, `Y`, and `Z` (which are hidden from view in the above picture), since pressing those squares has the same effect as pressing the squares opposite to them on the faces `X`, `Y`, `Z`.

```
data Face : Set where
  X Y Z : Face

Square : Set
Square = Face × Fin n × Fin n

face : Square → Face
face (f , _ , _) = f

coord : Square → Fin n × Fin n
coord (_ , i , j) = i , j
```

The effect of pressing each square can be described succinctly using the `Bool`-valued equality test on `Fin n` and Boolean operations. Pressing a square toggles a lamp just in case the projection of the lamp onto the `face` of the square coincides with the coordinate `coord` of the square.

```
lampOf : Square → Lamp
lampOf (X , i , j) = 0F , i , j
lampOf (Y , i , j) = i , 0F , j
lampOf (Z , i , j) = i , j , 0F

project : Face → Lamp → (Fin n × Fin n)
project X (i , j , k) = (j , k)
project Y (i , j , k) = (i , k)
project Z (i , j , k) = (i , j)

project-face : (s : Square) → project (face s) (lampOf s) ≡ coord s
project-face (X , p) = refl
project-face (Y , p) = refl
project-face (Z , p) = refl
```

```
press : Square → Configuration → Configuration
press (f , i , j) c l with (i₁ , j₁) ← project f l
  = (i ≡ᵇ i₁ ∧ j ≡ᵇ j₁) xor c l

pressList : List Square → Configuration → Configuration
pressList xs c = foldr press c xs

pressList-++ : (xs ys : List Square) →
  pressList (xs ++ ys) ≗ pressList xs ∘ pressList ys
pressList-++ xs ys c = foldr-++ press c xs ys
```

The question is when pressing some sequence of squares can bring a given configuration `c` to one in which all lamps are off. In other words, we are asked to find a necessary and sufficient condition for the following to hold:

```
∃ λ xs → AllOff (pressList xs c)
```

The first thing to notice is that no matter which configuration to start with, we can always turn off all lamps that are visible (i.e., all lamps one of whose coordinate is `0F`). This is done by visiting each of the three faces in turn, pressing all and only the squares that are lit.

```
turnOffSquare : Configuration → Square → List Square
turnOffSquare c s = if c (lampOf s) then [ s ] else []

turnOffSquare-property₁ : (c d : Configuration) (s : Square) →
  c (lampOf s) ≡ d (lampOf s) →
  pressList (turnOffSquare c s) d (lampOf s) ≡ false
turnOffSquare-property₁ c d (X , p) h with c (lampOf (X , p))
... | false = sym h
... | true = sym (cong not h)
turnOffSquare-property₁ c d (Y , p) h with c (lampOf (Y , p))
... | false = sym h
... | true = sym (cong not h)
turnOffSquare-property₁ c d (Z , p) h with c (lampOf (Z , p))
... | false = sym h
... | true = sym (cong not h)

turnOffSquare-property₂ : (c d : Configuration) (s : Square) (l : Lamp) →
  coord s ≠ project (face s) l →
  pressList (turnOffSquare c s) d l ≡ d l
turnOffSquare-property₂ c d (f , i , j) l h with c (lampOf (f , i , j))
... | false = refl
... | true with i ≟ proj₁ (project f l) | j ≟ proj₂ (project f l)
... | no ¬a | _ = refl
... | yes a | no ¬b = refl
... | yes a | yes b = ⊥-elim (h (cong₂ _,_ a b))

turnOffSquare-property₃ :
  (f : Face) (c d : Configuration) (xs : List (Fin n × Fin n)) (l : Lamp) →
  project f l ∉ xs →
  pressList (concatMap (λ p → turnOffSquare c (f , p)) xs) d l ≡ d l
turnOffSquare-property₃ f c d [] l h = refl
turnOffSquare-property₃ f c d ((i₁ , j₁) :: xs) l h = begin
  pressList (concatMap g ((i₁ , j₁) :: xs)) d l
    ≡⟨⟩
  pressList (g (i₁ , j₁) ++ concatMap g xs) d l
    ≡⟨ cong-app (pressList-++ (g (i₁ , j₁)) (concatMap g xs) d) l ⟩
  pressList (g (i₁ , j₁)) d′ l
    ≡⟨ turnOffSquare-property₂ c d′ (f , i₁ , j₁) l [2] ⟩
  d′ l
    ≡⟨ turnOffSquare-property₃ f c d xs l [1] ⟩
  d l
```

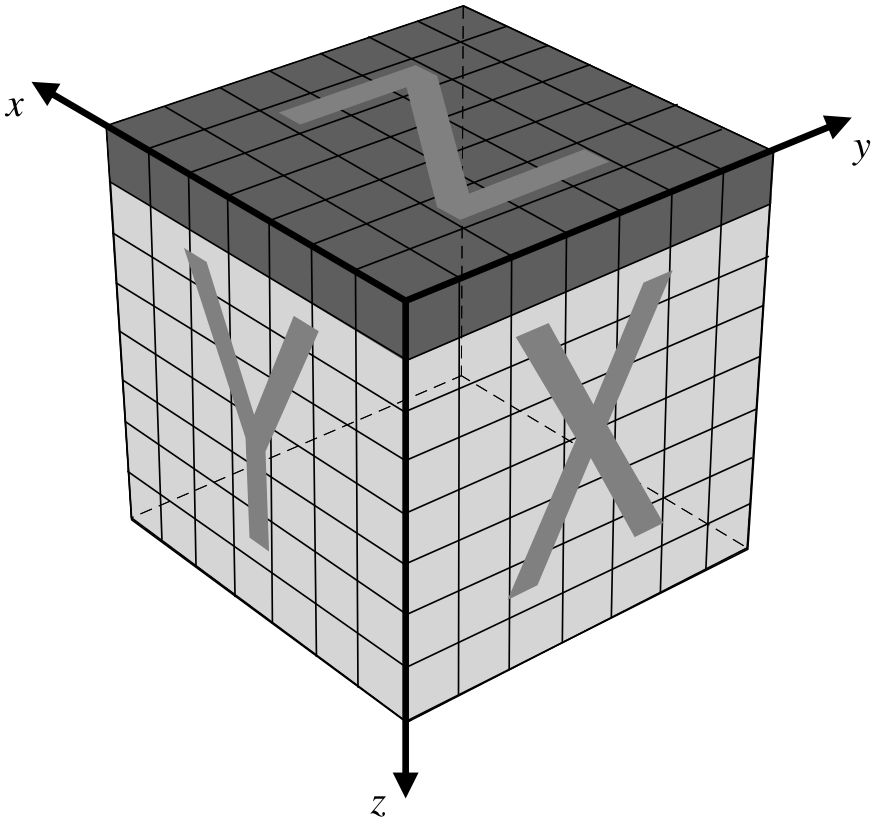
```
  where
  open ≡-Reasoning

  g : Fin n × Fin n → List Square
  g p = turnOffSquare c (f , p)

  i j : Fin n
  i = proj₁ (project f l)
  j = proj₂ (project f l)

  d′ : Configuration
  d′ = pressList (concatMap g xs) d

  [1] : (i , j) ∉ xs
  [1] h₁ = h (there h₁)

  [2] : (i₁ , j₁) ≢ (i , j)
  [2] refl = h (here refl)
```

`turnOffSquare c s` is either `[ s ]` or `[]`, depending on whether `s` is lit in the configuration `c`. We concatenate these lists for each square `s` on `f` to get `turnOffFace c f`.

```
enumerateFin : (m : ℕ) → List (Fin m)
enumerateFin zero = []
enumerateFin (suc m) = 0F :: map 1+ (enumerateFin m)

enumerateFin×Fin : (m : ℕ) → List (Fin m × Fin m)
enumerateFin×Fin m = cartesianProduct (enumerateFin m) (enumerateFin m)

turnOffFace : Configuration → Face → List Square
turnOffFace c f = concatMap (λ p → turnOffSquare c (f , p)) (enumerateFin×Fin n)
```

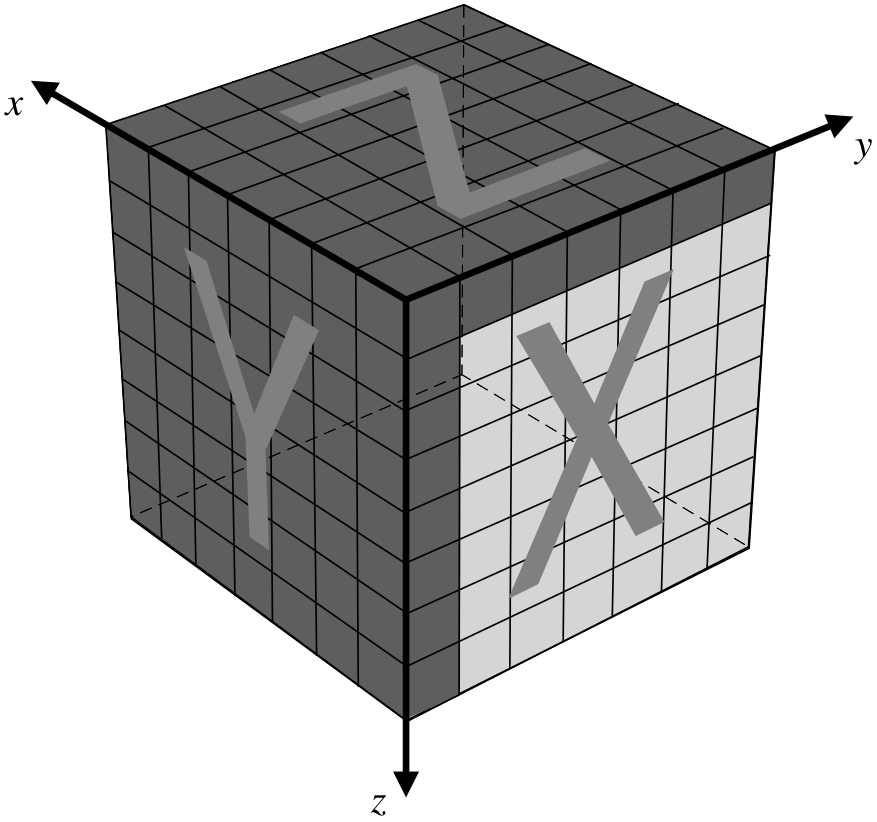
`turnOffFace c f` consists of all squares on the face `f` that are lit in the configuration `c`, listed in the lexicographic order.

Starting from the configuration `c` depicted in the earlier picture, pressing the squares in `turnOffFace c Z` (from right to left) brings us to a configuration `c₁` that looks like the following picture:



All lamps that are on the face `Z` are now off, which is indicated by the darker shade of gray. Other lamps may have changed states; some of them may be on and others off.

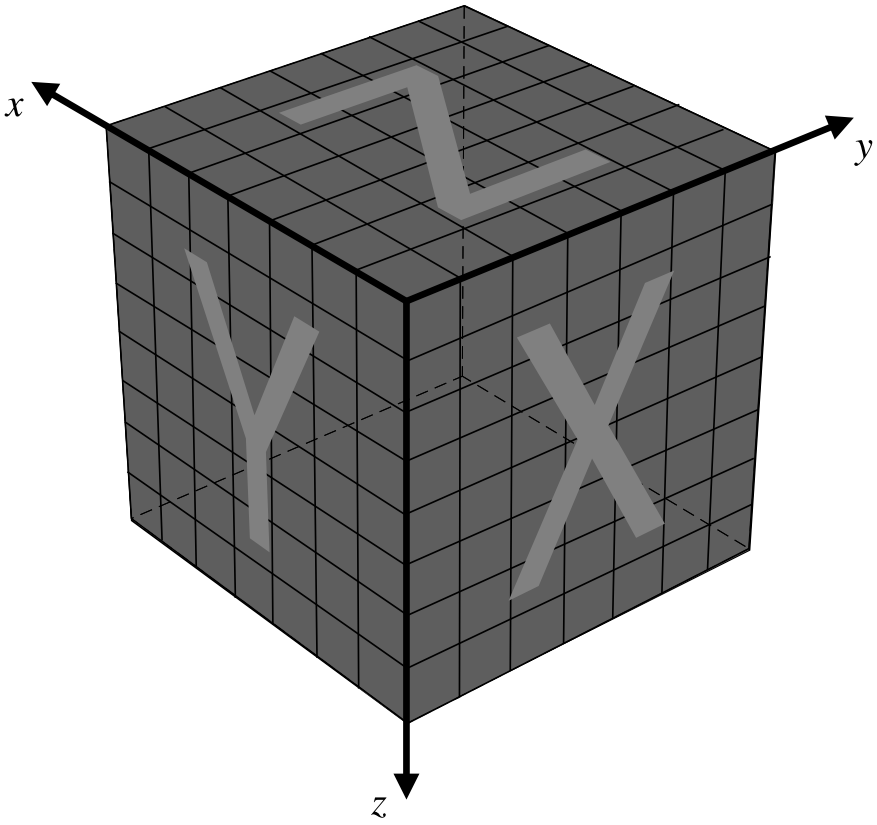
Pressing the squares in `turnOffFace` c_1 `Y` then brings us to the following configuration c_2:



Since the `Y` squares on the top row were already dark in c_1, they were not pressed, so all the lamps that are on the face `Z` remain off in c_2.

We can then press the squares in `turnOffFace` c_2 `X`, which brings us to a configuration c_3 in which all

lamps that are visible from our vantage point are off. (Of course, each of the invisible lamps may be on or off at this point.)



This is all intuitively trivial, but proving that a given configuration can always be transformed into one like the above picture in a proof assistant requires some work. There seems to be no obvious way to proceed that immediately suggests itself.

I choose to reason about each individual lamp independently of what happens to the other lamps. Note that for each `(i , j)`, pressing the squares in `turnOffFace c f` can be broken down into pressing some squares on `f` different from `(f , i , j)`, followed by possibly pressing `(f , i , j)`, followed by pressing some more squares on `f` different from `(f , i , j)`.

```
enumerateFin-split : (m : ℕ) (i : Fin m) →
  ∃₂ λ xs ys → (enumerateFin m ≡ xs ++ i :: ys) ×
    i ∉ xs × i ∉ ys
enumerateFin-split (suc m) 0F =
  [] , map 1+ (enumerateFin m) , refl , (λ ()) , [1] (enumerateFin m)
  where
  [1] : (xs : List (Fin m)) → 0F ∉ map 1+ xs
  [1] (x :: xs) (there h) = [1] xs h
enumerateFin-split (suc m) (1+ i)
  with xs , ys , eq , h₁ , h₂ ← enumerateFin-split m i =
    0F :: map 1+ xs , map 1+ ys , cong (0F ::_) [1] , [2] , [3]
  where
  open ≡-Reasoning

  [1] : map 1+ (enumerateFin m) ≡ map 1+ xs ++ 1+ i :: map 1+ ys
  [1] = begin
    map 1+ (enumerateFin m)       ≡⟨ cong (map 1+) eq ⟩
    map 1+ (xs ++ i :: ys)        ≡⟨ map-++ 1+ xs (i :: ys) ⟩
    map 1+ xs ++ 1+ i :: map 1+ ys ∎

  [2] : 1+ i ∉ 0F :: map 1+ xs
  [2] (there h) with y , y∈xs , refl ← ∈-map⁻ 1+ h = h₁ y∈xs

  [3] : 1+ i ∉ map 1+ ys
```

```
  [3] h with y , y∈ys , refl ← ∈-map⁻ 1+ h = h₂ y∈ys

enumerateFin×Fin-split : (m : ℕ) (p : Fin m × Fin m) →
  ∃₂ λ xs ys → (enumerateFin×Fin m ≡ xs ++ p :: ys) ×
    p ∉ xs × p ∉ ys
enumerateFin×Fin-split m (i , j)
  with xs₁ , ys₁ , eq₁ , h₁₁ , h₁₂ ← enumerateFin-split m i
  with xs₂ , ys₂ , eq₂ , h₂₁ , h₂₂ ← enumerateFin-split m j =
  xs , ys , [2] , [3] , [4]
  where
  open ≡-Reasoning

  xs = cartesianProduct xs₁ (enumerateFin m) ++ map (i ,_) xs₂
  ys = map (i ,_) ys₂ ++ cartesianProduct ys₁ (enumerateFin m)

  [1] : map (i ,_) (enumerateFin m) ≡ map (i ,_) xs₂ ++ (i , j) :: map (i ,_) ys₂
  [1] = begin
    (map (i ,_) (enumerateFin m)              ≡⟨ cong (map (i ,_)) eq₂ ⟩
    map (i ,_) (xs₂ ++ j :: ys₂)              ≡⟨ map-++ (i ,_) xs₂ (j :: ys₂) ⟩
    map (i ,_) xs₂ ++ map (i ,_) (j :: ys₂)   ≡⟨⟩
    map (i ,_) xs₂ ++ (i , j) :: map (i ,_) ys₂ ∎)

  [2] : enumerateFin×Fin m ≡ xs ++ (i , j) :: ys
  [2] = begin
    enumerateFin×Fin m ≡⟨⟩
    cartesianProduct (enumerateFin m) (enumerateFin m)
      ≡⟨ cong (λ ● → cartesianProduct ● (enumerateFin m)) eq₁ ⟩
    cartesianProduct (xs₁ ++ i :: ys₁) (enumerateFin m)
      ≡⟨ cartesianProductWith-distribʳ-++ _,_ xs₁ (i :: ys₁) (enumerateFin m) ⟩
    cartesianProduct xs₁ (enumerateFin m) ++
      cartesianProduct (i :: ys₁) (enumerateFin m)
      ≡⟨⟩
    cartesianProduct xs₁ (enumerateFin m) ++ map (i ,_) (enumerateFin m) ++
      cartesianProduct ys₁ (enumerateFin m)
      ≡⟨ cong (λ ● → cartesianProduct xs₁ (enumerateFin m) ++ ● ++
                cartesianProduct ys₁ (enumerateFin m)) [1] ⟩
    cartesianProduct xs₁ (enumerateFin m) ++ (map (i ,_) xs₂ ++
      (i , j) :: map (i ,_) ys₂) ++ cartesianProduct ys₁ (enumerateFin m)
      ≡⟨ cong (cartesianProduct xs₁ (enumerateFin m) ++_)
          (++-assoc (map (i ,_) xs₂) ((i , j) :: map (i ,_) ys₂)
            (cartesianProduct ys₁ (enumerateFin m))) ⟩
    cartesianProduct xs₁ (enumerateFin m) ++ map (i ,_) xs₂ ++
      (i , j) :: map (i ,_) ys₂ ++ cartesianProduct ys₁ (enumerateFin m)
      ≡⟨ sym (++-assoc (cartesianProduct xs₁ (enumerateFin m)) (map (i ,_) xs₂)
                ((i , j) :: map (i ,_) ys₂ ++ cartesianProduct ys₁ (enumerateFin m))) ⟩
    xs ++ (i , j) :: ys
      ∎

  [3] : (i , j) ∉ xs
  [3] h with ∈-++⁻ (cartesianProduct xs₁ (enumerateFin m)) h
  ... | inj₁ x
    with a , _ , a∈xs₁ , _ , refl ← ∈-cartesianProductWith⁻ _,_ xs₁ (enumerateFin m) x
    = h₁₁ a∈xs₁
  ... | inj₂ y with b , b∈xs₂ , refl ← ∈-map⁻ (i ,_) y = h₂₁ b∈xs₂

  [4] : (i , j) ∉ ys
  [4] h with ∈-++⁻ (map (i ,_) ys₂) h
  ... | inj₁ x with b , b∈ys₂ , refl ← ∈-map⁻ (i ,_) x = h₂₂ b∈ys₂
  ... | inj₂ y
    with a , _ , a∈ys₁ , _ , refl ← ∈-cartesianProductWith⁻ _,_ ys₁ (enumerateFin m) y
    = h₁₂ a∈ys₁

turnOffFace-split : (c : Configuration) (f : Face) (p : Fin n × Fin n) →
  ∃₂ λ xs ys →
    pressList (turnOffFace c f) ≗
```

```
      pressList (concatMap (λ q → turnOffSquare c (f , q)) xs) ∘
        (pressList (turnOffSquare c (f , p))) ∘
        (pressList (concatMap (λ q → turnOffSquare c (f , q)) ys)) ×
    p ∉ xs × p ∉ ys
turnOffFace-split c f p
  with xs , ys , eq , h₁ , h₂ ← enumerateFin×Fin-split n p
  = xs , ys , [2] , h₁ , h₂
  where
  open ≡-Reasoning

  g : (Fin n × Fin n) → List Square
  g q = turnOffSquare c (f , q)

  [1] : concatMap g (enumerateFin×Fin n) ≡ concatMap g xs ++ g p ++ concatMap g ys
  [1] = begin
    concatMap g (enumerateFin×Fin n)
      ≡⟨ cong (concatMap g) eq ⟩
    concatMap g (xs ++ p :: ys)
      ≡⟨ cong concat (map-++ g xs (p :: ys)) ⟩
    concat (map g xs ++ g p :: map g ys)
      ≡⟨ sym (concat-++ (map g xs) (g p :: map g ys)) ⟩
    concatMap g xs ++ g p ++ concatMap g ys
      ∎

  [2] : pressList (concatMap g (enumerateFin×Fin n)) ≗
        pressList (concatMap g xs) ∘ (pressList (g p)) ∘ (pressList (concatMap g ys))
  [2] c = begin
    pressList (concatMap g (enumerateFin×Fin n)) c
      ≡⟨ cong (λ ● → pressList ● c) [1] ⟩
    pressList (concatMap g xs ++ g p ++ concatMap g ys) c
      ≡⟨ pressList-++ (concatMap g xs) (g p ++ concatMap g ys) c ⟩
    pressList (concatMap g xs) (pressList (g p ++ concatMap g ys) c)
      ≡⟨ cong (pressList (concatMap g xs)) (pressList-++ (g p) (concatMap g ys) c) ⟩
    pressList (concatMap g xs) (pressList (g p) (pressList (concatMap g ys) c))
      ∎
```

This allows us to prove two important properties of `turnOffFace`. First, starting from the configuration `c`, pressing the squares in `turnOffFace c f` indeed darkens all squares on the face `f`.

```
turnOffFace-property₁ : (c : Configuration) (f : Face) (p : Fin n × Fin n) →
  pressList (turnOffFace c f) c (lampOf (f , p)) ≡ false
turnOffFace-property₁ c f p with xs , ys , eq , h₁ , h₂ ← turnOffFace-split c f p
  = trans (cong-app (eq c) (lampOf (f , p))) [6]
  where
  g : (Fin n × Fin n) → List Square
  g q = turnOffSquare c (f , q)

  c₁ c₂ c₃ : Configuration
  c₁ = pressList (concatMap g ys) c
  c₂ = pressList (g p) c₁
  c₃ = pressList (concatMap g xs) c₂

  [1] : project f (lampOf (f , p)) ≡ p
  [1] = project-face (f , p)

  [2] : project f (lampOf (f , p)) ∉ xs
  [2] rewrite [1] = h₁

  [3] : project f (lampOf (f , p)) ∉ ys
  [3] rewrite [1] = h₂

  [4] : c₁ (lampOf (f , p)) ≡ c (lampOf (f , p))
  [4] = turnOffSquare-property₃ f c c ys (lampOf (f , p)) [3]
```

```
  [5] : c₂ (lampOf (f , p)) ≡ false
  [5] = turnOffSquare-property₁ c c₁ (f , p) (sym [4])

  [6] : c₃ (lampOf (f , p)) ≡ false
  [6] = trans (turnOffSquare-property₃ f c c₂ xs (lampOf (f , p)) [2]) [5]
```

Second, if the projection of a lamp `l` onto the face `f` is a square that is already dark in the configuration `c`, then pressing the squares in `turnOffFace c f` does not affect the state of the lamp `l`.

```
turnOffFace-property₂ : (c : Configuration) (f : Face) (l : Lamp) →
  c (lampOf (f , project f l)) ≡ false →
  pressList (turnOffFace c f) c l ≡ c l
turnOffFace-property₂ c f l h
  with xs , ys , eq , h₁ , h₂ ← turnOffFace-split c f (project f l)
  = trans (cong-app (eq c) l) [4]
  where
  p : Fin n × Fin n
  p = project f l

  g : (Fin n × Fin n) → List Square
  g q = turnOffSquare c (f , q)

  c₁ c₂ c₃ : Configuration
  c₁ = pressList (concatMap g ys) c
  c₂ = pressList (g p) c₁
  c₃ = pressList (concatMap g xs) c₂

  [1] : c₁ l ≡ c l
  [1] = turnOffSquare-property₃ f c c ys l h₂

  [2] : g p ≡ []
  [2] rewrite h = refl

  [3] : c₂ ≡ c₁
  [3] rewrite [2] = refl

  [4] : c₃ l ≡ c l
  [4] rewrite [3] = trans (turnOffSquare-property₃ f c c₁ xs l h₁) [1]
```

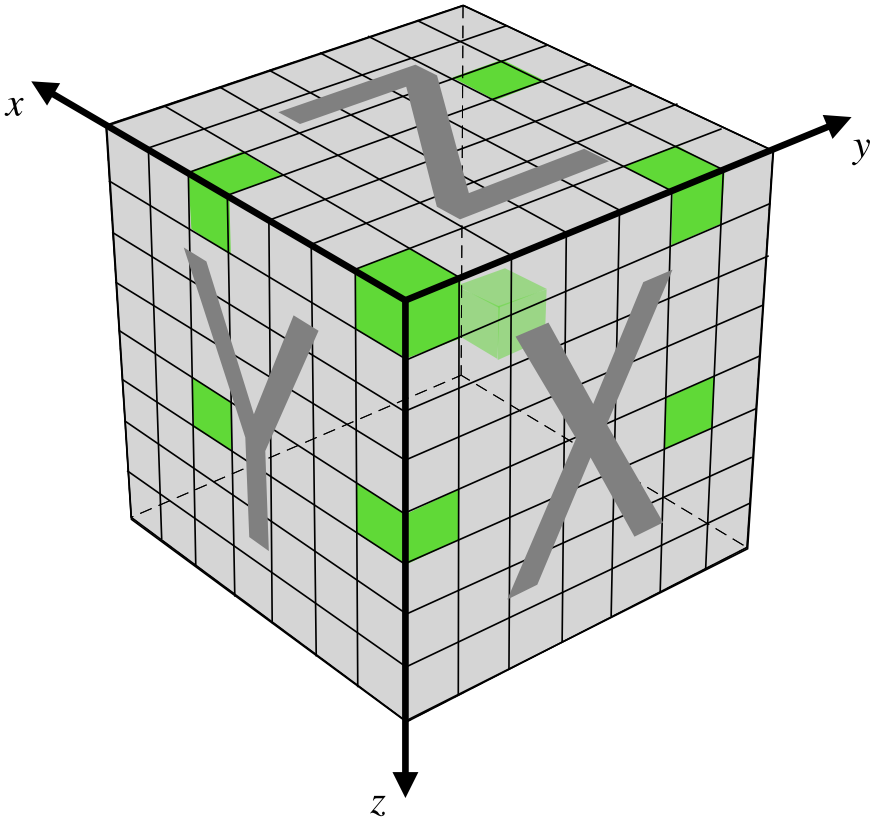
We are now ready to prove what we called “intuitively trivial” above.

```
turnOffAllFaces : Configuration → List Square
turnOffAllFaces c = turnOffFace c₂ X ++ turnOffFace c₁ Y ++ turnOffFace c Z
  where
  c₁ c₂ : Configuration
  c₁ = pressList (turnOffFace c Z) c
  c₂ = pressList (turnOffFace c₁ Y) c₁

AllOffX : Configuration → Set
AllOffX c = (j k : Fin n) → c (0F , j , k) ≡ false

AllOffY : Configuration → Set
AllOffY c = (i k : Fin n) → c (i , 0F , k) ≡ false

AllOffZ : Configuration → Set
AllOffZ c = (i j : Fin n) → c (i , j , 0F) ≡ false

AllOffXYZ : Configuration → Set
AllOffXYZ c = AllOffX c × AllOffY c × AllOffZ c

turnOffAllFaces-property : (c : Configuration) →
  AllOffXYZ (pressList (turnOffAllFaces c) c)
turnOffAllFaces-property c = [8]
  where
```

```
open ≡-Reasoning

c₁ c₂ c₃ : Configuration
c₁ = pressList (turnOffFace c Z) c
c₂ = pressList (turnOffFace c₁ Y) c₁
c₃ = pressList (turnOffFace c₂ X) c₂

[1] : pressList (turnOffAllFaces c) c ≡ c₃
[1] = begin
  pressList (turnOffAllFaces c) c
    ≡⟨⟩
  pressList (turnOffFace c₂ X ++ turnOffFace c₁ Y ++ turnOffFace c Z) c
    ≡⟨ pressList-++ (turnOffFace c₂ X) (turnOffFace c₁ Y ++ turnOffFace c Z) c ⟩
  pressList (turnOffFace c₂ X) (pressList (turnOffFace c₁ Y ++ turnOffFace c Z) c)
    ≡⟨ cong (pressList (turnOffFace c₂ X))
        (pressList-++ (turnOffFace c₁ Y) (turnOffFace c Z) c) ⟩
  c₃
    ∎

[2] : AllOffZ c₁
[2] i j = turnOffFace-property₁ c Z (i , j)

[3] : AllOffY c₂
[3] i k = turnOffFace-property₁ c₁ Y (i , k)

[4] : AllOffZ c₂
[4] i j = trans (turnOffFace-property₂ c₁ Y (i , j , 0F) ([2] i 0F)) ([2] i j)

[5] : AllOffX c₃
[5] j k = turnOffFace-property₁ c₂ X (j , k)

[6] : AllOffY c₃
[6] i k = trans (turnOffFace-property₂ c₂ X (i , 0F , k) ([3] 0F k)) ([3] i k)

[7] : AllOffZ c₃
[7] i j = trans (turnOffFace-property₂ c₂ X (i , j , 0F) ([4] 0F j)) ([4] i j)

[8] : AllOffXYZ (pressList (turnOffAllFaces c) c)
[8] rewrite [1] = [5] , [6] , [7]
```

So, starting from any configuration, we can turn off all lamps that are visible from our vantage point. In the resulting configuration (which we called `c₃`), some lamps that are invisible from our vantage point may be on. If so, is there any additional sequence of squares that we can press to turn all lamps off? This seems difficult because in order to toggle an invisible lamp, we must press a square on one of the faces `X`, `Y`, and `Z`, which necessarily toggles a visible lamp. We can prove that it is indeed impossible. Given an initial configuration, the states of the visible lamps in any reachable configuration completely determines the states of all other lamps.

For each invisible lamp `(1+ i , 1+ j , 1+ k)`, we associate seven visible lamps with it. These are the lamps we get by changing some or all of the coordinates of `(1+ i , 1+ j , 1+ k)` to `0F`. The following picture highlights the seven visible lamps associated with the lamp `(4F , 5F , 4F)`.



We note that the XOR of the state of the invisible lamp `(1+ i , 1+ j , 1+ k)` and the states of the seven visible lamps associated with it is an invariant, since pressing any square toggles either zero or exactly two of these eight lamps. This means that, given an initial configuration, the state of an invisible lamp in any reachable configuration is completely determined by the states of the seven visible lamps associated with it.

Given a configuration `c`, we refer to the function that associates with each invisible lamp the XOR of the states in `c` of the eight associated lamps as the *signature* of `c`. The signature of a configuration is invariant under any square-pressing. The signature of an all-off configuration is clearly the constant `false` function. Therefore, for a configuration `c` to be turned into an all-off configuration by pressing some sequence of squares, it is necessary and sufficient that `c` has the constant `false` function as its signature.

```
Signature : Set
Signature = (Fin n₁ × Fin n₁ × Fin n₁) → Bool

signature : Configuration → Signature
signature c (i , j , k) =
  c (0F , 0F , 0F) xor c (0F , 0F , 1+ k) xor c (0F , 1+ j , 0F) xor
  c (0F , 1+ j , 1+ k) xor c (1+ i , 0F , 0F) xor c (1+ i , 0F , 1+ k) xor
  c (1+ i , 1+ j , 0F) xor c (1+ i , 1+ j , 1+ k)

signature-AllOff : (c : Configuration) → AllOff c → signature c ≗ λ _ → false
signature-AllOff c h (i , j , k)
  rewrite h 0F 0F 0F
        | h 0F 0F (1+ k)
        | h 0F (1+ j) 0F
        | h 0F (1+ j) (1+ k)
        | h (1+ i) 0F 0F
        | h (1+ i) 0F (1+ k)
        | h (1+ i) (1+ j) 0F
        | h (1+ i) (1+ j) (1+ k) = refl

signature-press : (s : Square) (c : Configuration) →
  signature (press s c) ≗ signature c
signature-press (X , j , k) c (i₁ , j₁ , k₁)
  with j ≡ᵇ 1+ j₁ | k ≡ᵇ 1+ k₁ | j ≡ᵇ 0F | k ≡ᵇ 0F
```

```
... | false | false | false | false = refl
... | false | false | false | true = refl
... | false | false | true | false = refl
... | false | false | true | true = refl
... | false | true | false | false = refl
... | false | true | false | true = refl
... | false | true | true | false = refl
... | false | true | true | true = refl
... | true | false | false | false = refl
... | true | false | false | true = refl
... | true | false | true | false = refl
... | true | false | true | true = refl
... | true | true | false | false = refl
... | true | true | false | true = refl
... | true | true | true | false = refl
... | true | true | true | true = refl
signature-press (Y , i , k) c (i₁ , j₁ , k₁)
  with i ≡ᵇ 1+ i₁ | k ≡ᵇ 1+ k₁ | i ≡ᵇ 0F | k ≡ᵇ 0F
... | false | false | false | false = refl
... | false | false | false | true = refl
... | false | false | true | false = refl
... | false | false | true | true = refl
... | false | true | false | false = refl
... | false | true | false | true = refl
... | false | true | true | false = refl
... | false | true | true | true = refl
... | true | false | false | false = refl
... | true | false | false | true = refl
... | true | false | true | false = refl
... | true | false | true | true = refl
... | true | true | false | false = refl
... | true | true | false | true = refl
... | true | true | true | false = refl
... | true | true | true | true = refl
signature-press (Z , i , j) c (i₁ , j₁ , k₁)
  with i ≡ᵇ 1+ i₁ | j ≡ᵇ 1+ j₁ | i ≡ᵇ 0F | j ≡ᵇ 0F
... | false | false | false | false = refl
... | false | false | false | true = refl
... | false | false | true | false = refl
... | false | false | true | true = refl
... | false | true | false | false = refl
... | false | true | false | true = refl
... | false | true | true | false = refl
... | false | true | true | true = refl
... | true | false | false | false = refl
... | true | false | false | true = refl
... | true | false | true | false = refl
... | true | false | true | true = refl
... | true | true | false | false = refl
... | true | true | false | true = refl
... | true | true | true | false = refl
... | true | true | true | true = refl

signature-pressList : (xs : List Square) (c : Configuration) →
  signature (pressList xs c) ≗ signature c
signature-pressList [] c _ = refl
signature-pressList (x :: xs) c z = begin
  signature (pressList (x :: xs) c) z     ≡⟨⟩
  signature (press x (pressList xs c)) z ≡⟨ signature-press x (pressList xs c) z ⟩
  signature (pressList xs c) z           ≡⟨ signature-pressList xs c z ⟩
  signature c z                          ∎
  where open ≡-Reasoning

sufficiency-lemma₁ :
  (c : Configuration) → AllOffXYZ c → signature c ≗ (λ _ → false) → AllOff c
```

```
sufficiency-lemma₁ c (hx , hy , hz) h 0F j k = hx j k
sufficiency-lemma₁ c (hx , hy , hz) h (1+ i) 0F k = hy (1+ i) k
sufficiency-lemma₁ c (hx , hy , hz) h (1+ i) (1+ j) 0F = hz (1+ i) (1+ j)
sufficiency-lemma₁ c (hx , hy , hz) h (1+ i) (1+ j) (1+ k)
  = trans [1] (h (i , j , k))
  where
  [1] : c (1+ i , 1+ j , 1+ k) ≡ signature c (i , j , k)
  [1] rewrite hx 0F 0F | hx 0F (1+ k) | hx (1+ j) 0F | hx (1+ j) (1+ k)
              | hy (1+ i) 0F | hy (1+ i) (1+ k) | hz (1+ i) (1+ j)
    = refl

sufficiency : (c : Configuration) →
  signature c ≗ (λ _ → false) →
  ∃ λ xs → AllOff (pressList xs c)
sufficiency c h =
  turnOffAllFaces c , sufficiency-lemma₁ c₁ (turnOffAllFaces-property c) [1]
  where
  c₁ : Configuration
  c₁ = pressList (turnOffAllFaces c) c

  [1] : signature c₁ ≗ λ _ → false
  [1] z = trans (signature-pressList (turnOffAllFaces c) c z) (h z)

necessity : (c : Configuration) →
  (∃ λ xs → AllOff (pressList xs c)) → signature c ≗ (λ _ → false)
necessity c (xs , h) z =
  trans (sym (signature-pressList xs c z)) (signature-AllOff c₁ h z)
  where
  c₁ : Configuration
  c₁ = pressList xs c
```

Is this necessary and sufficient condition “as concise as possible”? The condition can be expressed as a Boolean formula which is a conjunction of the negations of $(n-1)^3$ XORs, each of which contains eight Boolean variables, so the total number of occurrences of Boolean variables in the formula is $8(n-1)^3$. The formula involves n^3 variables altogether, and of the 2^{n^3} rows of its truth table, exactly 2^{3n^2-3n+1} of them makes the formula true. The question is whether there is an equivalent formula which contains significantly fewer than $8(n-1)^3$ occurrences of Boolean variables. I see no easy way of finding one. We really need all n^3 variables, so as a function of n, the length of our formula is at most a constant multiple of the length of the shortest equivalent formula.