```
open import Relation.Binary.PropositionalEquality
open Relation.Binary.PropositionalEquality.≡-Reasoning

data ℕ : Set where
  zero : ℕ
  suc  : ℕ → ℕ

_+_ : ℕ → ℕ → ℕ
zero  + y = y
suc x + y = suc (x + y)

data Vec (A : Set) : ℕ → Set where
  nil  : Vec A zero
  cons : {n : _} → A → Vec A n → Vec A (suc n)

head : {A : Set} {n : ℕ} → Vec A (suc n) → A
head (cons x xs) = x

tail : {A : Set} {n : ℕ} → Vec A (suc n) → Vec A n
tail (cons x xs) = xs

_++_ : {A : Set} {n m : ℕ} → Vec A n → Vec A m → Vec A (n + m)
nil       ++ ys = ys
cons x xs ++ ys = cons x (xs ++ ys)

+-assoc : ∀ n m l → n + (m + l) ≡ (n + m) + l
+-assoc zero    m l = refl
+-assoc (suc n) m l = cong suc (+-assoc n m l)

+-suc : ∀ n m → suc (m + n) ≡ m + suc n
+-suc n zero    = refl
+-suc n (suc m) = cong suc (+-suc n m)

+-comm : ∀ n m → n + m ≡ m + n
+-comm zero zero    = refl
+-comm zero (suc m) = cong suc (+-comm zero m)
+-comm (suc n) m = begin
  suc (n + m)
    ≡⟨ cong suc (+-comm n m) ⟩
  suc (m + n)
    ≡⟨ +-suc n m ⟩
  m + suc n ∎
```