```agda
module lib where

infix 4 _≤_ _<_ _≡_
infixl 6 _+_ _∸_ _−_
infixl 7 _*_

data ℕ : Set where
  zero : ℕ
  suc : ℕ → ℕ
{-# BUILTIN NATURAL ℕ #-}

data ℤ : Set where
  pos : ℕ → ℤ
  negsuc : ℕ → ℤ
{-# BUILTIN INTEGER          ℤ #-}
{-# BUILTIN INTEGERPOS    pos #-}
{-# BUILTIN INTEGERNEGSUC negsuc #-}

_+_ : ℕ → ℕ → ℕ
zero + n = n
suc m + n = suc (m + n)
{-# BUILTIN NATPLUS _+_ #-}

-- Monus (a∸b = max{a-b, 0})
_∸_ : ℕ → ℕ → ℕ
m ∸ zero = m
zero ∸ suc n = zero
suc m ∸ suc n = m ∸ n
{-# BUILTIN NATMINUS _∸_ #-}

_*_ : ℕ → ℕ → ℕ
zero * n = zero
suc m * n = n + m * n
{-# BUILTIN NATTIMES _*_ #-}

-- Relations of natural numbers
data _≡_ {l} {A : Set l} (x : A) : A → Set l where
  refl : x ≡ x
{-# BUILTIN EQUALITY _≡_ #-}

sym : ∀ {l} {A : Set l} {x y : A} → x ≡ y → y ≡ x
sym refl = refl

cong : ∀ {l l′} {A : Set l} {B : Set l′} (f : A → B) {x y : A} → x ≡ y → f x ≡ f y
cong f refl = refl

sub : ∀ {l l′} {A : Set l} {x y : A} (P : A → Set l′) → x ≡ y → P x → P y
sub P refl px = px

trans : ∀ {l} {A : Set l} {x y z : A} → x ≡ y → y ≡ z → x ≡ z
trans refl refl = refl

-- n∸n 0 :   {n : ℕ} → n ∸ n   zero
-- n∸n 0 {zero} = refl
-- n∸n 0 {suc n} = n∸n 0 {n}

-- n+1 suc-n :   {n} → n + 1   suc n
-- n+1 suc-n {zero} = refl
-- n+1 suc-n {suc n} rewrite n+1 suc-n {n} = refl

+-identityʳ : ∀ {n} → n + zero ≡ n
+-identityʳ {zero} = refl
+-identityʳ {suc n} rewrite +-identityʳ {n} = refl
-- +-identityʳ {suc n} = cong suc (+-identityʳ {n})

+-sucʳ : ∀ {m n} → m + suc n ≡ suc (m + n)
+-sucʳ {zero} {n} = refl
+-sucʳ {suc m} {n} rewrite +-sucʳ {m} {n} = refl
-- +-sucʳ {suc m} {n} = cong suc (+-sucʳ {m})

+-comm : ∀ {m n} → m + n ≡ n + m
+-comm {m} {zero} = +-identityʳ
+-comm {m} {suc n}
  rewrite (+-sucʳ {m} {n}) | (+-comm {m} {n}) = refl
-- +-comm {m} {suc n} = trans +-sucʳ (cong suc (+-comm {m} {n

data _≤_ : ℕ → ℕ → Set where
  z≤n : ∀ {n : ℕ} → zero ≤ n
  s≤s : ∀ {m n : ℕ} → m ≤ n → suc m ≤ suc n

≤-irrelevant : ∀ {m n} → (p₁ p₂ : m ≤ n) → p₁ ≡ p₂
≤-irrelevant z≤n z≤n = refl
≤-irrelevant (s≤s p₁) (s≤s p₂) = cong s≤s (≤-irrelevant p₁ p₂)

-- inv-s s :   {m n : ℕ} → suc m   suc n → m   n
-- inv-s s (s s m n) = m n

≤-refl : ∀ {n : ℕ} → n ≤ n
≤-refl {zero} = z≤n
≤-refl {suc n} = s≤s ≤-refl

≤-trans : ∀ {m n p : ℕ} → m ≤ n → n ≤ p → m ≤ p
≤-trans z≤n _ = z≤n
≤-trans (s≤s m≤n) (s≤s n≤p) = s≤s (≤-trans m≤n n≤p)

n≤suc-n : ∀ {n : ℕ} → n ≤ suc n
n≤suc-n {zero} = z≤n
n≤suc-n {suc n} = s≤s n≤suc-n

-- m   n, p   n → p   m
m≡n,p≤n→p≤m : ∀ {p m n} → m ≡ n → p ≤ n → p ≤ m
m≡n,p≤n→p≤m m≡n p≤n rewrite sym m≡n = p≤n

-- n n+1 :   {n : ℕ} → n   n + 1
-- n n+1 = m n,p n→p m n+1 suc-n n suc-n

+→≤ : ∀ {m n : ℕ} → m ≤ m + n
+→≤ {zero} {n} = z≤n
+→≤ {suc m} {n} = s≤s +→≤

+→≤ʳ : ∀ {m n : ℕ} → m ≤ n + m
-- +→ ʳ {m} {zero} =  -refl {m}
+→≤ʳ {m} {n} = m≡n,p≤n→p≤m (+-comm {n} {m}) +→≤
-- +→ ʳ :   {m} →   {n} → m   n + m
-- +→ ʳ {m} {zero} =  -refl {m}
-- +→ ʳ {m} {n} = m n,p n→p m (+-comm {n} {m}) +→

data Order : ℕ → ℕ → Set where
  leq : ∀ {m n : ℕ} → m ≤ n → Order m n
  geq : ∀ {m n : ℕ} → n ≤ m → Order m n

≤-compare : ∀ {m n : ℕ} → Order m n
≤-compare {zero} {n} = leq z≤n
≤-compare {suc m} {zero} = geq z≤n
≤-compare {suc m} {suc n} with ≤-compare {m} {n}
... | leq m≤n = leq (s≤s m≤n)
... | geq n≤m = geq (s≤s n≤m)

-- ∸-  :   {m n} → m ∸ n   m
-- ∸-  {m} {zero} =  -refl
-- ∸-  {zero} {suc n} = z n
-- ∸-  {suc m} {suc n} =  -trans (∸-  {m} {n}) n suc_n

data _<_ : ℕ → ℕ → Set where
  z<s : ∀ {n : ℕ} → zero < suc n
  s<s : ∀ {m n : ℕ} → m < n → suc m < suc n

<→s≤ : ∀ {m n : ℕ} → m < n → suc m ≤ n
<→s≤ (z<s) = s≤s z≤n
<→s≤ (s<s m<n) = s≤s (<→s≤ m<n)

<→≤ : ∀ {m n : ℕ} → m < n → m ≤ n
<→≤ m<n = ≤-trans n≤suc-n (<→s≤ m<n)

<-trans : ∀ {m n p : ℕ} → m < n → n < p → m < p
<-trans z<s (s<s _) = z<s
<-trans (s<s m<n) (s<s n<p) = s<s (<-trans m<n n<p)

_−_ : (n : ℕ) → (m : ℕ) → (p : m ≤ n) → ℕ
(n − zero) (z≤n) = n
(suc n − suc m) (s≤s m≤n) = (n − m) m≤n

−-irrelevant : ∀ {n m} → (p₁ p₂ : m ≤ n) → (n − m) p₁ ≡ (n − m) p₂
−-irrelevant {n} {m} p₁ p₂ rewrite ≤-irrelevant p₁ p₂ = refl

-- n - m   n
−→≤ : ∀ {n m} → (m≤n : m ≤ n) → (n − m) m≤n ≤ n
−→≤ z≤n = ≤-refl
−→≤ (s≤s m≤n) = ≤-trans (−→≤ m≤n) n≤suc-n

n−n≡0 : ∀ {n} → (n − n) (≤-refl {n}) ≡ 0
n−n≡0 {zero} = refl
n−n≡0 {suc n} = n−n≡0 {n}

-- suc (n - m)   suc n - m
−-suc : ∀ {n m} → {m≤n : m ≤ n}
            → suc ((n − m) m≤n) ≡ (suc n − m) (≤-trans m≤n n≤suc-n)
−-suc {_} {zero} {z≤n} = refl
−-suc {suc n} {suc m} {s≤s m≤n} = −-suc {n} {m} {m≤n}

n−[n−m]≡m : ∀ {m n} → (m≤n : m ≤ n)
                  → (n − ((n − m) m≤n)) (−→≤ m≤n) ≡ m
n−[n−m]≡m {zero} {n} z≤n = n−n≡0 {n}
n−[n−m]≡m {suc m} {suc n} (s≤s m≤n) =
  trans (sym (−-suc {n} {(n − m) m≤n}))
            (cong suc (n−[n−m]≡m {m} {n} m≤n))

n+m−m≡n : ∀ {m n} → (n + m − m) (+→≤ʳ) ≡ n
n+m−m≡n {m} {zero} =
  trans (−-irrelevant {m} {m} +→≤ʳ ≤-refl) (n−n≡0 {m})
n+m−m≡n {m} {suc n} =
  trans
    (−-irrelevant {suc n + m} {m} +→≤ʳ (≤-trans +→≤ʳ n≤suc-n))
    (trans (sym (−-suc {n + m} {m}))
                  (cong suc (n+m−m≡n {m} {n})))

-- -- m   n → m - p   n - p
-- --monoʳ-  :   {p m n} → (p m : p   m) → (m n : m   n)
--                   → (m - p) p m   (n - p) ( -trans p m m n)
-- --monoʳ-  z n m n = m n
-- --monoʳ-  (s s p m) (s s m n) = --monoʳ-  p m m n

-- suc d   d → d   d - (d - (suc d))
suc-d≤d′→d≤d′−[d′−[suc-d]] : ∀ {d d′} → (δ₁≤δ₂ : suc d ≤ d′)
            → d ≤ ((d′ − ((d′ − (suc d)) δ₁≤δ₂)) (−→≤ δ₁≤δ₂))
suc-d≤d′→d≤d′−[d′−[suc-d]] δ₁≤δ₂ =
            m≡n,p≤n→p≤m (n−[n−m]≡m δ₁≤δ₂) n≤suc-n
```