```
module source where

open import lib

-- Operator precedence and associativity
infix 1 _≤:_
infix 2 _⟶_
infix 4 _⊢_
infix 4 _∈_
infixl 5 _,_
infixr 7 _⇒_

-- Types
data Type : Set where
  comm intexp intacc intvar : Type
  _⇒_ : Type → Type → Type

-- Subtype relation
data _≤:_ : Type → Type → Set where
  ≤:-refl : ∀{A} → A ≤: A
  ≤:-trans : ∀{A A’ A”} → A ≤: A’ → A’ ≤: A” → A ≤: A”
  ≤:-fn : ∀{A A’ B B’} → A’ ≤: A → B ≤: B’ → A ⇒ B ≤: A’ ⇒ B’

  var-≤:-exp : intvar ≤: intexp
  var-≤:-acc : intvar ≤: intacc

-- Contexts
data Context : Set where
  · : Context
  _,_ : Context → Type → Context

-- Variables and the lookup judgement
data _∈_ : Type → Context → Set where
  Zero : ∀{Γ A} → A ∈ Γ , A
  Suc : ∀{Γ A B} → B ∈ Γ → B ∈ Γ , A

-- Terms and the typing judgement
data _⊢_ : Context → Type → Set where
  Var : ∀{Γ A} → A ∈ Γ → Γ ⊢ A

  -- subtyping
  Sub : ∀{Γ A B} → Γ ⊢ A → A ≤: B → Γ ⊢ B

  -- lambda function and application
  Lambda : ∀{Γ A B} → Γ , A ⊢ B → Γ ⊢ A ⇒ B
  App : ∀{Γ A B} → Γ ⊢ A ⇒ B → Γ ⊢ A → Γ ⊢ B

  -- command
  Skip : ∀{Γ} → Γ ⊢ comm
  Seq : ∀{Γ} → Γ ⊢ comm → Γ ⊢ comm → Γ ⊢ comm
  NewVar : ∀{Γ} → Γ , intvar ⊢ comm → Γ ⊢ comm
  Assign : ∀{Γ} → Γ ⊢ intacc → Γ ⊢ intexp → Γ ⊢ comm

  -- intexp
  Lit : ∀{Γ} → ℤ → Γ ⊢ intexp
  Neg : ∀{Γ} → Γ ⊢ intexp → Γ ⊢ intexp
  Plus : ∀{Γ} → Γ ⊢ intexp → Γ ⊢ intexp → Γ ⊢ intexp


-- Operational semantics
data Value : ∀{Γ A} → Γ ⊢ A → Set where
  V-Lambda : ∀{Γ A B} {F : Γ , A ⊢ B} → Value (Lambda {Γ} F)
  V-Lit : ∀{Γ} {i : ℤ} → Value (Lit {Γ} i)
  V-Skip : ∀{Γ} → Value (Skip {Γ})

-- Renaming
ext : ∀{Γ Δ} → (∀{A} → A ∈ Γ → A ∈ Δ)
               → (∀{A B} → B ∈ Γ , A → B ∈ Δ , A)
ext ρ Zero = Zero
ext ρ (Suc x) = Suc (ρ x)

rename : ∀{Γ Δ} → (∀{A} → A ∈ Γ → A ∈ Δ)
                  → (∀{A} → Γ ⊢ A → Δ ⊢ A)
rename ρ (Var A∈Γ) = Var (ρ A∈Γ)
rename ρ (Lambda Γ,A⊢B) = Lambda (rename (ext ρ) Γ,A⊢B)
rename ρ (Sub Γ⊢A A≤:B) = Sub (rename ρ Γ⊢A) A≤:B
rename ρ (App Γ⊢A Γ⊢B) = App (rename ρ Γ⊢A) (rename ρ Γ⊢B)
rename ρ Skip = Skip
rename ρ (Seq Γ⊢c₁ Γ⊢c₂) = Seq (rename ρ Γ⊢c₁) (rename ρ Γ⊢c₂)
rename ρ (NewVar Γ⊢c) = NewVar (rename (ext ρ) Γ⊢c)
rename ρ (Assign Γ⊢i Γ⊢e) = Assign (rename ρ Γ⊢i) (rename ρ Γ⊢e)
rename ρ (Lit Γ⊢i) = Lit Γ⊢i
rename ρ (Neg Γ⊢i) = Neg (rename ρ Γ⊢i)
rename ρ (Plus Γ⊢i₁ Γ⊢i₂) = Plus (rename ρ Γ⊢i₁) (rename ρ Γ⊢i₂)

-- Simultaneous substitution
exts : ∀{Γ Δ} → (∀{A} → A ∈ Γ → Δ ⊢ A)
                → (∀{A B} → B ∈ Γ , A → Δ , A ⊢ B)
exts σ Zero = Var Zero
exts σ (Suc x) = rename Suc (σ x)

subst : ∀{Γ Δ} → (∀{A} → A ∈ Γ → Δ ⊢ A)
                 → (∀{A} → Γ ⊢ A → Δ ⊢ A)
subst σ (Var A∈Γ) = σ A∈Γ
subst σ (Sub Γ⊢A A≤:B) = Sub (subst σ Γ⊢A) A≤:B
subst σ (Lambda Γ,A⊢B) = Lambda (subst (exts σ) Γ,A⊢B)
subst σ (App Γ⊢A Γ⊢B) = App (subst σ Γ⊢A) (subst σ Γ⊢B)
subst σ Skip = Skip
subst σ (Seq Γ⊢c₁ Γ⊢c₂) = Seq (subst σ Γ⊢c₁) (subst σ Γ⊢c₂)
subst σ (NewVar Γ⊢c) = NewVar (subst (exts σ) Γ⊢c)
subst σ (Assign Γ⊢i Γ⊢e) = Assign (subst σ Γ⊢i) (subst σ Γ⊢e)
subst σ (Lit Γ⊢i) = Lit Γ⊢i
subst σ (Neg Γ⊢i) = Neg (subst σ Γ⊢i)
subst σ (Plus Γ⊢i₁ Γ⊢i₂) = Plus (subst σ Γ⊢i₁) (subst σ Γ⊢i₂)

-- Single substitution
_[_] : ∀{Γ A B} → Γ , B ⊢ A → Γ ⊢ B → Γ ⊢ A
_[_] {Γ} {A} {B} N M = subst {Γ , B} {Γ} σ {A} N
  where
  σ : ∀ {A} → A ∈ Γ , B → Γ ⊢ A
  σ Zero = M
  σ (Suc x) = Var x

-- Reduction
data _⟶_ : ∀{Γ A} → (Γ ⊢ A) → (Γ ⊢ A) → Set where
  App-cong₁ : ∀{Γ A B} {F F’ : Γ ⊢ A ⇒ B} {E : Γ ⊢ A}
                        → F ⟶ F’ → App F E ⟶ App F’ E
  App-cong₂ : ∀{Γ A B} {V : Γ ⊢ A ⇒ B} {E E’ : Γ ⊢ A}
                        → Value V → E ⟶ E’ → App V E ⟶ App V E’
  Lambda-β : ∀{Γ A B} {F : Γ , A ⊢ B} {V : Γ ⊢ A}
                        → Value V → App (Lambda F) V ⟶ F [ V ]
```