

```

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 : Type
  intexp : Type
  intacc : Type
  intvar : Type
  _⇒_ : Type → Type → Type

-- 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

  -- 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

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

data Value : ∀{Γ A} → Γ ⊢ A → Set where
  V-Lit : ∀{Γ} {i : ℤ} → Value (Lit {Γ} i)

  V-Lambda : ∀{Γ A B} {F : Γ , A ⊢ B} → Value (Lambda {Γ} F)

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

rename : ∀{Γ Δ} → (∀{A} → A ∈ Γ → A ∈ Δ) → (∀{A} → Γ ⊢ A → Δ ⊢ A)
rename (Var x) = Var ( x)
rename (Lambda F) = Lambda (rename (ext ) F)
rename (App F E) = App (rename F) (rename E)
rename Skip = Skip
rename (Seq c c) = Seq (rename c) (rename c)
rename (Lit i) = Lit i
rename (Neg I) = Neg (rename I)
rename (Plus I I) = Plus (rename I) (rename I)

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

subst : ∀{Γ Δ} → (∀{A} → A ∈ Γ → Δ ⊢ A) → (∀{A} → Γ ⊢ A → Δ ⊢ A)
subst (Var x) = x
subst (Lambda F) = Lambda (subst (exts ) F)
subst (App F E) = App (subst F) (subst E)
subst Skip = Skip
subst (Seq c c) = Seq (subst c) (subst c)
subst (Lit i) = Lit i
subst (Neg I) = Neg (subst I)
subst (Plus I I) = Plus (subst I) (subst I)

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

-- Reduction
data _→_ : ∀{Γ A} → (Γ ⊢ A) → (Γ ⊢ A) → Set where
  App-cong : ∀{Γ A B} {F F : Γ ⊢ A ⇒ B} {E : Γ ⊢ A} → F → F → E → E → Value V → Value V
  App-cong : ∀{Γ A B} {V : Γ ⊢ A ⇒ B} {E E : Γ ⊢ A} → Value V → Value V
  Lambda- : ∀{Γ A B} {F : Γ , A ⊢ B} {V : Γ ⊢ A} → Value V → App

-- Subtype relation
data _≤_ : Type → Type → Set where
  ≤:-refl : ∀{T} → T ≤: T
  ≤:-trans : ∀{T T' T''} → T ≤: T' → T' ≤: T'' → T ≤: T''
  ≤:-fn : ∀{T T' T'' T'''} → T ≤: T' → T' ≤: T'' → T'' ≤: T''' → T ≤: T'''

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

```