```
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 , 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 x) = Var (ρ x)
rename ρ (Lambda F) = Lambda (rename (ext ρ) F)
rename ρ (App F E) = App (rename ρ F) (rename ρ E)
rename ρ Skip = Skip
rename ρ (Seq c₁ c₂) = Seq (rename ρ c₁) (rename ρ c₂)
rename ρ (Lit i) = Lit i
rename ρ (Neg I) = Neg (rename ρ I)
rename ρ (Plus I₁ I₂) = Plus (rename ρ I₁) (rename ρ 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 x) = σ x
subst σ (Lambda F) = Lambda (subst (exts σ) F)
subst σ (App F E) = App (subst σ F) (subst σ E)
subst σ Skip = Skip
subst σ (Seq c₁ c₂) = Seq (subst σ c₁) (subst σ c₂)
subst σ (Lit i) = Lit i
subst σ (Neg I) = Neg (subst σ I)
subst σ (Plus I₁ I₂) = Plus (subst σ I₁) (subst σ 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 ]

-- Subtype relation
data _≤:_ : Type → Type → Set where
    ≤:-refl : ∀{T} → T ≤: T
    ≤:-trans : ∀{T T' T''} → T ≤: T' → T' ≤: T'' → T ≤: T''
    ≤:-fn : ∀{T₁ T₁′ T₂ T₂′} → T₁′ ≤: T₁ → T₂ ≤: T₂′ → T₁ ⇒ T₂ ≤: T₁′ ⇒ T₂′

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