```
module compiler where

open import source
open import target
open import lib


infixr 1 _⇒ₛ_
infixl 2 _×_

-- Product and projection function
data _×_ (A B : Set) : Set where
    _,_ : A → B → A × B

π₁ : ∀ {A B} → A × B → A
π₁ (a , _) = a

π₂ : ∀ {A B} → A × B → B
π₂ (_ , b) = b

--  Type Interpretation
Compl : SD → Set
Compl sd = I sd

_×ₛ_ : (SD → Set) → (SD → Set) → SD → Set
(P ×ₛ Q) sd = P sd × Q sd

_⇒ₛ_ : (SD → Set) → (SD → Set) → SD → Set
(P ⇒ₛ Q) sd = ∀{sd'} → (sd ≤ₛ sd') → P sd' → Q sd'

Intcompl : SD → Set
Intcompl = R ⇒ₛ Compl


⟦_⟧ty : Type → SD → Set
⟦ comm ⟧ty = Compl ⇒ₛ Compl
⟦ intexp ⟧ty = Intcompl ⇒ₛ Compl
⟦ intacc ⟧ty = Compl ⇒ₛ Intcompl
⟦ intvar ⟧ty = ⟦ intexp ⟧ty ×ₛ ⟦ intacc ⟧ty
⟦ θ₁ ⇒ θ₂ ⟧ty = ⟦ θ₁ ⟧ty ⇒ₛ ⟦ θ₂ ⟧ty

-- Unit type for empty context
data ∅ : Set where
    unit : ∅

-- Context Interpretation
⟦_⟧ctx : Context → SD → Set
⟦ · ⟧ctx _ = ∅
⟦ Γ , A ⟧ctx sd = ⟦ Γ ⟧ctx sd × ⟦ A ⟧ty sd

get-var : ∀ {Γ A sd} → A ∈ Γ → ⟦ Γ ⟧ctx sd → ⟦ A ⟧ty sd
get-var Z     (_ , a) = a
```

```
get-var (S x) (γ , _) = get-var x γ


-- get-num : ∀ {Γ} → (e : Γ ⊢ source.ℕ) → target.ℕ
-- get-num Zero = target.zero
-- get-num (Suc m) = target.suc (get-num m)

fmap-⇒ : ∀ {P Q sd sd'} → (P ⇒ₛ Q) sd → sd ≤ₛ sd' → (P ⇒ₛ Q) sd'
fmap-⇒ θ p p' x = θ (≤ₛ-trans p p') x

fmap-Compl : ∀ {sd sd'} → Compl sd → sd ≤ₛ sd' → Compl sd'
fmap-Compl {sd} c (<-f f<f') = popto sd (<-f f<f') c
fmap-Compl {⟨ _ , d ⟩} {⟨ _ , d' ⟩} c (≤-d d≤d') = adjustdisp_dec (≤→Fin (-
→≤ {d'} {≤→Fin d≤d'}) ) (sub I {!   !} c)

-- fmap-× : ∀ {P Q sd sd'} → (P ×ₛ Q) sd → sd ≤ₛ sd' → (P ×ₛ Q) sd'

fmap-A : ∀ {A sd sd'} → ⟦ A ⟧ty sd → sd ≤ₛ sd' → ⟦ A ⟧ty sd'
fmap-A {comm} c p i = {!   !}
fmap-A {intexp} = {!   !}
fmap-A {intacc} = {!   !}
fmap-A {intvar} = {!   !}
fmap-A {A ⇒ B} = fmap-⇒ {⟦ A ⟧ty} {⟦ B ⟧ty}

fmap-Γ : ∀ {Γ sd sd'} → ⟦ Γ ⟧ctx sd → sd ≤ₛ sd' → ⟦ Γ ⟧ctx sd'
fmap-Γ {·} unit _ = unit
fmap-Γ {Γ , A} (γ , a) p = fmap-Γ γ p , fmap-A {A} a p


⟦_⟧ : ∀ {Γ A} → (e : Γ ⊢ A) → (sd : SD) → ⟦ Γ ⟧ctx sd → ⟦ A ⟧ty sd
⟦ Var x ⟧ _ γ = get-var x γ
⟦ Lambda f ⟧ sd γ {sd' = sd'} p a = ⟦ f ⟧ sd' (fmap-Γ γ p , a)
⟦ App f e ⟧ sd γ = ⟦ f ⟧ sd γ (≤-d ≤-refl) (⟦ e ⟧ sd γ)
⟦ Skip ⟧ _ _ _ γ = γ
⟦ Seq c₁ c₂ ⟧ sd γ sd' p = ⟦ c₁ ⟧ sd γ sd' (⟦ c₂ ⟧ sd γ sd' p)
⟦ Lit i ⟧ _ _ _ κ = κ ≤ₛ-refl (r-s (s-lit i))
⟦ Neg e ⟧ sd γ p κ = {!   !}
⟦ Plus e e₁ ⟧ _ γ = {!   !}
```