```
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

⟦_⟧var : ∀ {Γ A sd} → A ∈ Γ → ⟦ Γ ⟧ctx sd → ⟦ A ⟧ty sd
⟦ Zero ⟧var (_ , a) = a
⟦ Suc b ⟧var (γ , _) = ⟦ b ⟧var γ

⟦_⟧sub : ∀ {A A′ sd} → A ≤: A′ → ⟦ A ⟧ty sd → ⟦ A′ ⟧ty sd
⟦ ≤:-refl ⟧sub a = a
⟦ ≤:-trans A≤:A′ A′≤:A″ ⟧sub a = ⟦ A′≤:A″ ⟧sub (⟦ A≤:A′ ⟧sub a)
⟦ ≤:-fn A≤:A′ B′≤:B ⟧sub a =
    λ sd≤ssd′ a′ → ⟦ B′≤:B ⟧sub (a sd≤ssd′ (⟦ A≤:A′ ⟧sub a′))
⟦ var-≤:-exp ⟧sub (exp , acc) = exp
⟦ var-≤:-acc ⟧sub (exp , acc) = acc


-- Functorial mapping

fmap-I : ∀ {sd sd′} → I sd → sd ≤s sd′ → I sd′
fmap-I {sd} c (<-f f<f′) = popto sd (<-f f<f′) c
fmap-I {⟨ f , d ⟩} {⟨ f , d′ ⟩} c (≤-d d≤d′) =
    adjustdisp-dec ((d′ − d) d≤d′) (−→≤ d≤d′)
        (I-sub {n = (d′ − d) d≤d′} (n−[n−m]≡m d≤d′) c)

fmap-L : ∀ {sd sd′} → L sd → sd ≤s sd′ → L sd′
fmap-L (l-var sdv sdv≤ssd) sd≤ssd′ = l-var sdv (≤s-trans sdv≤ssd sd≤ssd′)
fmap-L (l-sbrs) _ = l-sbrs

fmap-S : ∀ {sd sd′} → S sd → sd ≤s sd′ → S sd′
fmap-S (s-l l) sd≤ssd′ = s-l (fmap-L l sd≤ssd′)
fmap-S (s-lit lit) _ = s-lit lit


fmap-⇒ : ∀ {P Q sd sd′} → (P ⇒s Q) sd → sd ≤s sd′ → (P ⇒s Q) sd′
fmap-⇒ P⇒Q sd≤ssd′ sd′≤ssd″ p = P⇒Q (≤s-trans sd≤ssd′ sd′≤ssd″) p

fmap-ty : ∀ {A sd sd′} → ⟦ A ⟧ty sd → sd ≤s sd′ → ⟦ A ⟧ty sd′
fmap-ty {comm} = fmap-⇒ {Compl} {Compl}
fmap-ty {intexp} = fmap-⇒ {Intcompl} {Compl}
fmap-ty {intacc} = fmap-⇒ {Compl} {Intcompl}
fmap-ty {intvar} ( exp , acc ) sd≤ssd′ =
    ( fmap-ty {intexp} exp sd≤ssd′ , fmap-ty {intacc} acc sd≤ssd′ )
fmap-ty {A ⇒ B} = fmap-⇒ {⟦ A ⟧ty} {⟦ B ⟧ty}

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


sd≤ssd′→sd≤ssd′−s[d′−[suc-d]] : ∀ {sd sd′} → sd ≤s sd′
    → (δ1≤δ2 : suc (SD.d sd) ≤ SD.d sd′)
    → sd ≤s ((sd′ −s ((SD.d sd′ − (suc (SD.d sd))) δ1≤δ2)) (−→≤ δ1≤δ2))
sd≤ssd′→sd≤ssd′−s[d′−[suc-d]] {⟨ f , _ ⟩} {⟨ f′ , _ ⟩} (<-f f<f′) _
    = <-f f<f′
sd≤ssd′→sd≤ssd′−s[d′−[suc-d]] {⟨ f , d ⟩} {⟨ f , d′ ⟩} (≤-d d≤d′) δ1≤δ2
    = ≤-d (suc-d≤d′→d≤d′−[d′−[suc-d]] δ1≤δ2)

new-intvar : ∀ sd → ⟦ intvar ⟧ty sd
new-intvar sd = ( exp , acc )
    where
        exp : ⟦ intexp ⟧ty sd
        exp sd≤ssd′ β = β ≤s-refl (r-s (s-l (l-var sd sd≤ssd′)))
        acc : ⟦ intacc ⟧ty sd
        acc {sd′ = sd′} sd≤ssd′ κ (≤-d {d = d′} {d′ = d″} d′≤d″) r
            = assign-dec
                ((d″ − d′) d′≤d″) (−→≤ d′≤d″)
                (l-var sd
                  (sub-sd≤s
                    (−s≡ {n≤d′ = −→≤ d′≤d″} (n−[n−m]≡m d′≤d″))
                    sd≤ssd′))
                r
                (l-sub {n = (d″ − d′) d′≤d″}(n−[n−m]≡m d′≤d″) κ)
        acc {sd′ = sd′} sd≤ssd′ κ (<-f f<f′) r
            = assign-inc 0 (l-var _ ≤s-refl) r (fmap-I κ (<-f f<f′))


assign : (sd : SD) → (sd′ : SD) → (S ⇒s Compl) sd
            → sd ≤s sd′ → R sd′ → I sd′
assign ⟨ f , d ⟩ ⟨ f′ , d′ ⟩ β sd≤ssd′ r with (≤-compare {suc d} {d′})
... | leq δ1≤δ2
        = assign-dec
            ((d′ − (suc d)) δ1≤δ2) (−→≤ δ1≤δ2)
              (l-var ⟨ f , d ⟩
                (sd≤ssd′→sd≤ssd′−s[d′−[suc-d]] sd≤ssd′ δ1≤δ2))
              r
              (β ((sd≤ssd′→sd≤ssd′−s[d′−[suc-d]] sd≤ssd′ δ1≤δ2))
                (s-l (l-var ⟨ f , d ⟩
                  ((sd≤ssd′→sd≤ssd′−s[d′−[suc-d]] sd≤ssd′ δ1≤δ2)))))


... | geq δ2≤δ1 = assign-inc (((suc d) − d′) δ2≤δ1)
                    (l-var ⟨ f , d ⟩ (≤s-trans sd≤ssd′ +s→≤s)) r
                    (β ((≤s-trans sd≤ssd′ +s→≤s))
                      (s-l (l-var ⟨ f , d ⟩ ((≤s-trans sd≤ssd′ +s→≤s)))))

use-temp : ∀ {sd sd′} → (S ⇒s Compl) sd → sd ≤s sd′ → R sd′ → I sd′
use-temp β sd≤ssd′ (r-s s) = β sd≤ssd′ s
use-temp {sd} {sd′} β sd≤ssd′ (r-unary uop s) =
    assign sd sd′ β sd≤ssd′ (r-unary uop s)
use-temp {sd} {sd′} β sd≤ssd′ (r-binary s1 bop s2) =
    assign sd sd′ β sd≤ssd′ (r-binary s1 bop s2)


⟦_⟧ : ∀ {Γ A} → Γ ⊢ A → (sd : SD) → ⟦ Γ ⟧ctx sd → ⟦ A ⟧ty sd
⟦ Var a ⟧ sd γ = ⟦ a ⟧var γ

⟦ Sub a A≤:B ⟧ sd γ = ⟦ A≤:B ⟧sub (⟦ a ⟧ sd γ)

⟦ Lambda f ⟧ sd γ {sd′ = sd′} sd≤ssd′ a = ⟦ f ⟧ sd′ (fmap-ctx γ sd≤ssd′ , a)

⟦ App f e ⟧ sd γ = ⟦ f ⟧ sd γ (≤-d ≤-refl) (⟦ e ⟧ sd γ)

⟦ Skip ⟧ sd γ sd≤ssd′ κ = κ

⟦ Seq c1 c2 ⟧ sd γ sd≤ssd′ κ = ⟦ c1 ⟧ sd γ sd≤ssd′ (⟦ c2 ⟧ sd γ sd≤ssd′ κ)

⟦ NewVar c ⟧ sd γ {sd′ = sd′} sd≤ssd′ κ =
    assign-inc 1
      (l-var sd′ (≤-d +→≤))
      (r-s (s-lit (pos 0)))
      (⟦ c ⟧
          (sd′ +s 1)
          (fmap-ctx {Γ = _ , intvar}
              ((fmap-ctx γ sd≤ssd′ , new-intvar sd′))
              (+s→≤s {sd′} {1}))
          ≤s-refl
          (adjustdisp-dec 1 +→≤′
              (l-sub {d′ = SD.d sd′ + 1} {n = 1}
                  (n+m−m≡n {m = 1}) κ)))

⟦ Assign a e ⟧ sd γ sd≤ssd′ κ = ⟦ e ⟧ sd γ sd≤ssd′ (⟦ a ⟧ sd γ sd≤ssd′ κ)

⟦ Lit i ⟧ sd γ sd≤ssd′ β = β ≤s-refl (r-s (s-lit i))

⟦ Neg e ⟧ sd γ sd≤ssd′ β =
    ⟦ e ⟧ sd γ sd≤ssd′ (use-temp λ sd≤ssd′ s → β sd≤ssd′ (r-unary UNeg s))

⟦ Plus e1 e2 ⟧ sd γ sd≤ssd′ β =
    ⟦ e1 ⟧ sd γ sd≤ssd′
        (use-temp (λ sd′≤ssd″ s1 → ⟦ e2 ⟧ sd γ (≤s-trans sd≤ssd′ sd′≤ssd″)
            (use-temp (λ sd″≤ssd‴ s2 → β (≤s-trans sd≤ssd′ (≤s-trans sd′≤ssd″ sd″≤ssd‴))
                (r-binary (fmap-S s1 sd″≤ssd‴) BPlus s2)))))

compile-closed : · ⊢ comm → I ⟨ 0 , 0 ⟩
compile-closed t = ⟦ t ⟧ ⟨ 0 , 0 ⟩ unit ≤s-refl stop
```