```agda
{-# OPTIONS --allow-unsolved-metas #-}

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


-- Functorality
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 {⟨ f , d ⟩} {⟨ f , d' ⟩} c (≤-d d≤d') =
    adjustdisp-dec ((d' − d) d≤d') (−→≤ d≤d')
        (l-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-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'
                    (≤-d (m≡n,p≤n→p≤m (n−[n−m]≡m d'≤d") ≤-refl)))
                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 = {!  !}
        -- a {sd' = sd'} sd≤ₛsd' κ (≤-d {d = d'} {d' = d''} d'
        --       (l-var sd' (≤-d (m≡n,p≤n→p≤m (n-[n-m]≡m d'≤d'

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
                +→≤ʳ
                (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' κ = κ ≤ₛ-refl (r-s (s-lit i))

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

⟦ Plus e₁ e₂ ⟧ sd γ p κ =
    ⟦ e₁ ⟧ sd γ p (use-temp (λ p' s₁ → ⟦ e₂ ⟧ sd γ (≤ₛ-trans p p')
                        (use-temp (λ p" s₂ → κ (≤ₛ-trans p' p")
                          (r-binary (fmap-S s₁ p") BPlus s₂)))))

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