

PCF.index

November 30, 2025

Contents

1	PCF.Checks	2
2	PCF.Constants	4
3	PCF.Domain-Notation	5
4	PCF.Environments	7
5	PCF.Terms	8
6	PCF.Types	9
7	PCF.Variables	10
8	PCF.index	11

1 PCF.Checks

```
{-# OPTIONS --rewriting --confluence-check --lossy-unification #-}
open import Agda.Builtin.Equality
open import Agda.Builtin.Equality.Rewrite

module PCF.Checks where

open import Data.Bool.Base
open import Agda.Builtin.Nat
open import Relation.Binary.PropositionalEquality.Core
  using ( _ ≡ _ ; refl ; cong-app )

open import PCF.Domain-Notation
open import PCF.Types
open import PCF.Constants
open import PCF.Variables
open import PCF.Environments
open import PCF.Terms

{-# REWRITE fix-fix elim-SHARP-η elim-SHARP-⊥ true-cond false-cond #-}

-- Constants
pattern N n    = L (k n)
pattern succ   = L +1'
pattern pred⊥  = L -1'
pattern if     = L ⊃i
pattern Y      = L Y
pattern Z      = L Z

-- Variables
f = var 0  $\iota$ 
g = var 1 ( $\iota \Rightarrow \iota$ )
h = var 2 ( $\iota \Rightarrow \iota \Rightarrow \iota$ )
a = var 3  $\iota$ 
b = var 4  $\iota$ 

-- Arithmetic
check-41+1 :  $\mathcal{A}' \llbracket \text{succ } \sqcup N\ 41 \rrbracket \rho \perp \equiv \eta\ 42$ 
check-41+1 = refl

check-43-1 :  $\mathcal{A}' \llbracket \text{pred } \perp \sqcup N\ 43 \rrbracket \rho \perp \equiv \eta\ 42$ 
check-43-1 = refl

-- Binding
check-id :  $\mathcal{A}' \llbracket (\bar{\lambda} a \sqcup V\ a) \sqcup N\ 42 \rrbracket \rho \perp \equiv \eta\ 42$ 
check-id = refl

check-k :  $\mathcal{A}' \llbracket (\bar{\lambda} a \sqcup \bar{\lambda} b \sqcup V\ a) \sqcup N\ 42 \sqcup N\ 41 \rrbracket \rho \perp \equiv \eta\ 42$ 
check-k = refl

check-ki :  $\mathcal{A}' \llbracket (\bar{\lambda} a \sqcup \bar{\lambda} b \sqcup V\ b) \sqcup N\ 41 \sqcup N\ 42 \rrbracket \rho \perp \equiv \eta\ 42$ 
check-ki = refl
```

```

check-suc-41 :  $\mathcal{A}' \llbracket (\bar{\lambda} a \sqcup (\text{succ} \sqcup V a)) \sqcup N 41 \rrbracket \rho \perp \equiv \eta 42$ 
check-suc-41 = refl

check-pred-42 :  $\mathcal{A}' \llbracket (\bar{\lambda} a \sqcup (\text{pred} \sqcup V a)) \sqcup N 43 \rrbracket \rho \perp \equiv \eta 42$ 
check-pred-42 = refl

check-if-zero :  $\mathcal{A}' \llbracket \text{if} \sqcup (Z \sqcup N 0) \sqcup N 42 \sqcup N 0 \rrbracket \rho \perp \equiv \eta 42$ 
check-if-zero = refl

check-if-nonzero :  $\mathcal{A}' \llbracket \text{if} \sqcup (Z \sqcup N 42) \sqcup N 0 \sqcup N 42 \rrbracket \rho \perp \equiv \eta 42$ 
check-if-nonzero = refl

-- fix ( $\lambda f. 42$ )  $\equiv 42$ 
check-fix-const :
   $\mathcal{A}' \llbracket Y \sqcup (\bar{\lambda} f \sqcup N 42) \rrbracket \rho \perp$ 
   $\equiv \eta 42$ 
check-fix-const = fix-fix ( $\lambda x \rightarrow \eta 42$ )

-- fix ( $\lambda g. \lambda a. 42$ ) 2  $\equiv 42$ 
check-fix-lambda :
   $\mathcal{A}' \llbracket Y \sqcup (\bar{\lambda} g \sqcup \bar{\lambda} a \sqcup N 42) \sqcup N 2 \rrbracket \rho \perp$ 
   $\equiv \eta 42$ 
check-fix-lambda = refl

-- fix ( $\lambda g. \lambda a. \text{ifz } a \text{ then } 42 \text{ else } g (\text{pred } a)$ ) 5  $\equiv 42$ 
check-countdown :
   $\mathcal{A}' \llbracket Y \sqcup (\bar{\lambda} g \sqcup \bar{\lambda} a \sqcup$ 
     $(\text{if} \sqcup (Z \sqcup V a) \sqcup N 42 \sqcup (V g \sqcup (\text{pred} \sqcup V a))))$ 
     $\sqcup N 5$ 
   $\rrbracket \rho \perp$ 
   $\equiv \eta 42$ 
check-countdown = refl

-- fix ( $\lambda h. \lambda a. \lambda b. \text{ifz } a \text{ then } b \text{ else } h (\text{pred } a) (\text{succ } b)$ ) 2 40  $\equiv 42$ 
check-sum-42 :
   $\mathcal{A}' \llbracket (Y \sqcup (\bar{\lambda} h \sqcup \bar{\lambda} a \sqcup \bar{\lambda} b \sqcup$ 
     $(\text{if} \sqcup (Z \sqcup V a) \sqcup V b \sqcup (V h \sqcup (\text{pred} \sqcup V a) \sqcup (\text{succ} \sqcup V b))))$ 
     $\sqcup N 2 \sqcup N 40$ 
   $\rrbracket \rho \perp$ 
   $\equiv \eta 42$ 
check-sum-42 = refl
-- Exponential in first arg?

```

2 PCF.Constants

```

{-# OPTIONS --rewriting --confluence-check --lossy-unification #-}

module PCF.Constants where

open import Data.Bool.Base
using (Bool; true; false; if _ then _ else _)
open import Agda.Builtin.Nat
using (Nat; _ + _; _ - _; _ == _)

open import PCF.Domain-Notation
using (⟦ _ ⟧; η; _ SHARP; fix; ⊥; _ → _ , _ )
open import PCF.Types
using (Types; o; ι; _ ⇒ _; σ; D)

-- Syntax

data L : Types → Set where
  tt  : L o
  ff  : L o
  ⊃i : L (o ⇒ ι ⇒ ι ⇒ ι)
  ⊃o : L (o ⇒ o ⇒ o ⇒ o)
  Y   : {σ : Types} → L ((σ ⇒ σ) ⇒ σ)
  k   : (n : Nat) → L ι
  +1' : L (ι ⇒ ι)
  -1' : L (ι ⇒ ι)
  Z   : L (ι ⇒ o)

variable c : L σ

-- Semantics

A[ _ ] : L σ → ⟦ D σ ⟧

A[ tt ] = η true
A[ ff ] = η false
A[ ⊃i ] = _ → _ , _
A[ ⊃o ] = _ → _ , _
A[ Y ] = fix
A[ k n ] = η n
A[ +1' ] = (λ n → η (n + 1)) SHARP
A[ -1' ] = (λ n → if n == 0 then ⊥ else η (n - 1)) SHARP
A[ Z ] = (λ n → η (n == 0)) SHARP

```

3 PCF.Domain-Notation

```

{-# OPTIONS --rewriting --confluence-check --lossy-unification #-}
open import Agda.Builtin.Equality
open import Agda.Builtin.Equality.Rewrite

module PCF.Domain-Notation where

open import Relation.Binary.PropositionalEquality.Core
  using ( _ ≡ _ ) public

-----

-- Domains

postulate
  Domain : Set1
  ⟨⟨ _ ⟩⟩ : Domain → Set

variable
  D E : Domain
  P : Set
  d1 d2 : Set

postulate
  ⊥ : ⟨⟨ D ⟩⟩ -- bottom element

-----

-- Function domains

postulate
  _ →c _ : Domain → Domain → Domain -- assume continuous
  _ →s _ : Set → Domain → Domain    -- always continuous
  dom-cts : ⟨⟨ D →c E ⟩⟩ ≡ (⟨⟨ D ⟩⟩ → ⟨⟨ E ⟩⟩)
  set-cts  : ⟨⟨ P →s E ⟩⟩ ≡ (P → ⟨⟨ E ⟩⟩)

{-# REWRITE dom-cts set-cts #-}

postulate
  fix : ⟨⟨ (D →c D) →c D ⟩⟩ -- fixed point of endofunction

-- Properties
  fix-fix : (f : ⟨⟨ D →c D ⟩⟩) → fix f ≡ f (fix f)

-- Flat domains

postulate
  _ + ⊥ : Set → Domain -- lifted set
  η : ⟨⟨ P →s P + ⊥ ⟩⟩ -- inclusion
  _ SHARP : ⟨⟨ (P →s D) →c P + ⊥ →c D ⟩⟩ -- Kleisli extension

-- Properties
  elim-SHARP-η : (f : ⟨⟨ P →s D ⟩⟩) (p : P) → (f SHARP) (η p) ≡ f p
  elim-SHARP-⊥ : (f : ⟨⟨ P →s D ⟩⟩) → (f SHARP) ⊥ ≡ ⊥

```

```

-- McCarthy conditional

--  $t \longrightarrow d_1, d_2 : \ll D \gg$  ( $t : \text{Bool} + \perp$  ;  $d_1, d_2 : \ll D \gg$ )

open import Data.Bool.Base
using (Bool; true; false; if _ then _ else _) public

postulate
   $\_ \longrightarrow \_, \_ : \ll \text{Bool} + \perp \rightarrow^c D \rightarrow^c D \rightarrow^c D \gg$  -- McCarthy conditional

-- Properties
true-cond   :  $\{d_1\ d_2 : \ll D \gg\} \rightarrow (\eta\ \text{true} \longrightarrow d_1, d_2) \equiv d_1$ 
false-cond  :  $\{d_1\ d_2 : \ll D \gg\} \rightarrow (\eta\ \text{false} \longrightarrow d_1, d_2) \equiv d_2$ 
bottom-cond :  $\{d_1\ d_2 : \ll D \gg\} \rightarrow (\perp \longrightarrow d_1, d_2) \equiv \perp$ 

infixr 0    $\_ \rightarrow^c \_$ 
infixr 0    $\_ \rightarrow^s \_$ 
infix 10    $\_ + \perp$ 
infixr 20   $\_ \longrightarrow \_, \_$ 

```

4 PCF.Environments

```

{-# OPTIONS --rewriting --confluence-check --lossy-unification #-}

module PCF.Environments where

open import Data.Bool.Base
  using (Bool; if _ then _ else _)
open import Data.Maybe.Base
  using (Maybe; just; nothing)
open import Agda.Builtin.Nat
  using (Nat; _ == _)
open import Relation.Binary.PropositionalEquality.Core
  using ( _ ≡ _ ; refl; trans; cong)

open import PCF.Domain-Notation
  using (⟦ _ ⟧; ⊥)
open import PCF.Types
  using (Types; ⋈; o; _ ⇒ _; D)
open import PCF.Variables
  using (V; var; Env)

-- ρ⊥ is the initial environment

ρ⊥ : Env
ρ⊥ α = ⊥

-- (ρ [ x / α ]) α' = x when α and α' are identical, otherwise ρ α'

_ [ _ / _ ] : {σ : Types} → Env → ⟦ D σ ⟧ → V σ → Env
ρ [ x / α ] = λ α' → h ρ x α α' (α ==V α') where

  h : {σ τ : Types} → Env → ⟦ D σ ⟧ → V σ → V τ → Maybe (σ ≡ τ) → ⟦ D τ ⟧
  h ρ x α α' (just refl) = x
  h ρ x α α' nothing    = ρ α'

  _ ==T _ : (σ τ : Types) → Maybe (σ ≡ τ)
  (σ ⇒ τ) ==T (σ' ⇒ τ') = f (σ ==T σ') (τ ==T τ') where
    f : Maybe (σ ≡ σ') → Maybe (τ ≡ τ') → Maybe ((σ ⇒ τ) ≡ (σ' ⇒ τ'))
    f = λ { (just p) (just q) → just (trans (cong ( _ ⇒ τ) p) (cong (σ' ⇒ _) q))
          ; _ _ → nothing }

  ⋈ ==T ⋈ = just refl
  o ==T o = just refl
  _ ==T _ = nothing

  _ ==V _ : {σ τ : Types} → V σ → V τ → Maybe (σ ≡ τ)
  var i σ ==V var i' τ =
    if i == i' then σ ==T τ else nothing

```

5 PCF.Terms

```

{-# OPTIONS --rewriting --confluence-check #-}

module PCF.Terms where

open import PCF.Domain-Notation
  using (⟨⟨ _ ⟩⟩; _ →c _; _ →s _)
open import PCF.Types
  using (Types; _ ⇒ _; σ; D)
open import PCF.Constants
  using (L; A[_]; c)
open import PCF.Variables
  using (V; Env; _[_])
open import PCF.Environments
  using (_[_ / _])

-- Syntax

data Terms : Types → Set where
  V      : {σ : Types} → V σ → Terms σ      -- variables
  L      : {σ : Types} → L σ → Terms σ      -- constants
  _ ⊔ _  : {σ τ : Types} → Terms (σ ⇒ τ) → Terms σ → Terms τ -- application
  λ _ ⊔ _ : {σ τ : Types} → V σ → Terms τ → Terms (σ ⇒ τ) -- λ-abstraction

variable M N : Terms σ
infixl 20 _ ⊔ _

-- Semantics

A'[_] : Terms σ → ⟨⟨ Env →s D σ ⟩⟩

A'[_] V α      ] ρ = ρ [ α ]
A'[_] L c      ] ρ = A [ c ]
A'[_] M ⊔ N     ] ρ = A'[_] M ] ρ (A'[_] N ] ρ)
A'[_] λ α ⊔ M   ] ρ = λ x → A'[_] M ] (ρ [ x / α ])

```


6 PCF.Types

```
{-# OPTIONS --rewriting --confluence-check #-}

module PCF.Types where

open import Data.Bool.Base
using (Bool)
open import Agda.Builtin.Nat
using (Nat)

open import PCF.Domain-Notation
using (Domain;  $\ll \_ \gg$ ;  $\_ \rightarrow^c \_$ ;  $\_ + \perp$ )

-- Syntax

data Types : Set where
   $\iota$       : Types           -- natural numbers
   $\circ$       : Types           -- Boolean truthvalues
   $\_ \Rightarrow \_$  : Types  $\rightarrow$  Types  $\rightarrow$  Types -- functions

variable  $\sigma \tau$  : Types

infixr 1  $\_ \Rightarrow \_$ 

-- Semantics  $\mathcal{D}$ 

 $\mathcal{D}$  : Types  $\rightarrow$  Domain

 $\mathcal{D} \iota$       =  $\text{Nat} + \perp$ 
 $\mathcal{D} \circ$       =  $\text{Bool} + \perp$ 
 $\mathcal{D} (\sigma \Rightarrow \tau)$  =  $\mathcal{D} \sigma \rightarrow^c \mathcal{D} \tau$ 

variable  $x \ y \ z$  :  $\ll \mathcal{D} \sigma \gg$ 
```

7 PCF.Variables

```
{-# OPTIONS --rewriting --confluence-check #-}

module PCF.Variables where

open import Agda.Builtin.Nat
using (Nat)

open import PCF.Domain-Notation
using ((⟦ _ ⟧); _ →c _; _ →s _)
open import PCF.Types
using (Types; σ;  $\mathcal{D}$ )

-- Syntax

data  $\mathcal{V}$  : Types → Set where
  var : Nat → (σ : Types) →  $\mathcal{V}$  σ

variable α :  $\mathcal{V}$  σ

-- Environments

Env = {σ : Types} →  $\mathcal{V}$  σ → ⟦  $\mathcal{D}$  σ ⟧

variable ρ : Env

-- Semantics

_ [ ] : ⟦ Env →s  $\mathcal{V}$  σ →s  $\mathcal{D}$  σ ⟧

ρ [ α ] = ρ α
```

8 PCF.index

```
{-# OPTIONS --rewriting --confluence-check #-}  
  
module PCF.index where  
  
import PCF.Domain-Notation  
import PCF.Types  
import PCF.Constants  
import PCF.Variables  
import PCF.Environments  
import PCF.Terms  
import PCF.Checks
```