

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- $\eta$  elim-SHARP- $\perp$  true-cond false-cond #-}

-- Constants
pattern  $N n = L (k n)$ 
pattern  $\text{succ} = L +1'$ 
pattern  $\text{pred}\perp = L -1'$ 
pattern  $\text{if } b = L \triangleright_i$ 
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 :  $\mathcal{A}'[\text{succ} \sqcup N 41] \rho \perp \equiv \eta 42$ 
check-41+1 = refl

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

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

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

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

```

check-suc-41 : A'[(\bar{a} \sqcup (\text{succ} \sqcup V a)) \sqcup N 41] \rho\perp \equiv \eta 42
check-suc-41 = refl

check-pred-42 : A'[(\bar{a} \sqcup (\text{pred}\perp \sqcup V a)) \sqcup N 43] \rho\perp \equiv \eta 42
check-pred-42 = refl

check-if-zero : A'[if \sqcup (Z \sqcup N 0) \sqcup N 42 \sqcup N 0] \rho\perp \equiv \eta 42
check-if-zero = refl

check-if-nonzero : A'[if \sqcup (Z \sqcup N 42) \sqcup N 0 \sqcup N 42] \rho\perp \equiv \eta 42
check-if-nonzero = refl

-- fix (\lambda f. 42) \equiv 42
check-fix-const :
  A'[Y \sqcup (\bar{f} \sqcup N 42)] \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 :
  A'[Y \sqcup (\bar{g} \sqcup \bar{a} \sqcup N 42) \sqcup N 2] \rho\perp
  \equiv \eta 42
check-fix-lambda = refl

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

-- fix (\lambda h. \lambda a. \lambda b. ifz a then b else h (pred a) (succ b)) 2 40 \equiv 42
check-sum-42 :
  A'[(Y \sqcup (\bar{h} \sqcup \bar{a} \sqcup \bar{b} \sqcup
    (if \sqcup (Z \sqcup V a) \sqcup V b \sqcup (V h \sqcup (\text{pred}\perp \sqcup V a) \sqcup (\text{succ} \sqcup V b))))))
    \sqcup N 2 \sqcup N 40
  ] \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; ℓ; _ ⇒ _ ; σ; D)

-- Syntax

data L : Types → Set where
  tt  : L o
  ff  : L o
  ∃ᵢ  : L (o ⇒ ℓ ⇒ ℓ ⇒ ℓ)
  ∃ₒ  : L (o ⇒ o ⇒ o ⇒ o)
  Y   : {σ : Types} → L ((σ ⇒ σ) ⇒ σ)
  k   : (n : Nat) → L ℓ
  +1' : L (ℓ ⇒ ℓ)
  -1' : L (ℓ ⇒ ℓ)
  Z   : L (ℓ ⇒ 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 → d1 , d2 : ⟨⟨ D ⟩⟩ (t : Bool +⊥ ; d1, d2 : ⟨⟨ D ⟩⟩)

open import Data.Bool.Base
  using (Bool; true; false; if_ _ then_ _ else_ _) public

postulate
  _ → _ , _ : ⟨⟨ Bool +⊥ →c D →c D →c D ⟩⟩ -- McCarthy conditional

  -- Properties
  true-cond    : {d1 d2 : ⟨⟨ D ⟩⟩} → (η true → d1 , d2)      ≡ d1
  false-cond   : {d1 d2 : ⟨⟨ D ⟩⟩} → (η false → d1 , d2) ≡ d2
  bottom-cond  : {d1 d2 : ⟨⟨ D ⟩⟩} → (⊥ → d1 , d2)      ≡ ⊥

infixr 0    _ →c _
infixr 0    _ →s _
infix 10   _ _ +⊥
infixr 20   _ → _ , _

```

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; ℓ; o; _⇒_ ; ℐ)
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 → ⟨⟨ ℐ σ ⟩⟩ → V σ → Env
ρ [x/α] = λ α' → h ρ × α α' (α ==V α') where
  h : {σ τ : Types} → Env → ⟨⟨ ℐ σ ⟩⟩ → V σ → V τ → Maybe (σ ≡ τ) → ⟨⟨ ℐ τ ⟩⟩
  h ρ × α α' (just refl) = x
  h ρ × 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 }

ℓ ==T ℓ = 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 (⟨⟨_⟩⟩; _ $\rightarrow^c$ _ ; _ $\rightarrow^s$ _)
open import PCF.Types
using (Types; _ $\Rightarrow$ _ ;  $\sigma$ ;  $\mathcal{D}$ )
open import PCF.Constants
using ( $\mathcal{L}$ ;  $A[\_]$ ; c)
open import PCF.Variables
using ( $\mathcal{V}$ ; Env; _ $[ \ ]$ )
open import PCF.Environments
using (_[/_])

-- Syntax

data Terms : Types  $\rightarrow$  Set where
   $V$  : { $\sigma$  : Types}  $\rightarrow$   $\mathcal{V}$   $\sigma$   $\rightarrow$  Terms  $\sigma$  -- variables
   $L$  : { $\sigma$  : Types}  $\rightarrow$   $\mathcal{L}$   $\sigma$   $\rightarrow$  Terms  $\sigma$  -- constants
   $_\sqcup_$  : { $\sigma \tau$  : Types}  $\rightarrow$  Terms ( $\sigma \Rightarrow \tau$ )  $\rightarrow$  Terms  $\sigma$   $\rightarrow$  Terms  $\tau$  -- application
   $\bar{\lambda}_\sqcup_$  : { $\sigma \tau$  : Types}  $\rightarrow$   $\mathcal{V}$   $\sigma$   $\rightarrow$  Terms  $\tau$   $\rightarrow$  Terms ( $\sigma \Rightarrow \tau$ ) --  $\lambda$ -abstraction

variable M N : Terms  $\sigma$ 
infixl 20  $_\sqcup_$ 

-- Semantics

 $\mathcal{A}'[\_]$  : Terms  $\sigma$   $\rightarrow$  ⟨⟨ Env  $\rightarrow^s$   $\mathcal{D}$   $\sigma$  ⟩⟩

 $\mathcal{A}'[V \alpha]$   $\rho$  =  $\rho[\alpha]$ 
 $\mathcal{A}'[L c]$   $\rho$  =  $\mathcal{A}[c]$ 
 $\mathcal{A}'[M \sqcup N]$   $\rho$  =  $\mathcal{A}'[M] \rho (\mathcal{A}'[N] \rho)$ 
 $\mathcal{A}'[\bar{\lambda} \alpha \sqcup M]$   $\rho$  =  $\lambda x \rightarrow \mathcal{A}'[M](\rho[x/\alpha])$ 
```

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; ⟨⟨ _ ⟩⟩; _ →c _ ; _ +⊥)

-- Syntax

data Types : Set where
  ℓ      : Types           -- natural numbers
  o      : Types           -- Boolean truthvalues
  _ ⇒ _ : Types → Types → Types -- functions

variable σ τ : Types

infixr 1 _ ⇒ _

-- Semantics D

D : Types → Domain

D ℓ      = Nat +⊥
D o      = Bool +⊥
D (σ ⇒ τ) = D σ →c D τ

variable x y z : ⟨⟨ D σ ⟩⟩
```

7 PCF.Variables

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

module PCF.Variables where

open import Agda.Builtin.Nat
using (Nat)

open import PCF.Domain-Notation
using (⟨⟨_⟩⟩; _ $\rightarrow^c$ _; _ $\rightarrow^s$ _)
open import PCF.Types
using (Types;  $\sigma$ ;  $\mathcal{D}$ )

-- Syntax

data  $\mathcal{V}$  : Types  $\rightarrow$  Set where
  var : Nat  $\rightarrow$  ( $\sigma$  : Types)  $\rightarrow$   $\mathcal{V}$   $\sigma$ 

variable  $\alpha$  :  $\mathcal{V}$   $\sigma$ 

-- Environments

Env = { $\sigma$  : Types}  $\rightarrow$   $\mathcal{V}$   $\sigma$   $\rightarrow$  ⟨⟨  $\mathcal{D}$   $\sigma$  ⟩⟩

variable  $\rho$  : Env

-- Semantics

_⟨_⟩ : ⟨⟨ Env  $\rightarrow^s$   $\mathcal{V}$   $\sigma$   $\rightarrow^s$   $\mathcal{D}$   $\sigma$  ⟩⟩
 $\rho$  ⟨ $\alpha$ ⟩ =  $\rho$   $\alpha$ 
```

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
```