

PCF.index

November 20, 2025

Contents

1	PCF.Checks	2
2	PCF.Constants	5
3	PCF.Domain-Notation	6
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 #-}
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

fix-app : ∀ {P D} (f : (P → D) → (P → D)) (p : P) →
  fix f p ≡ f (fix f) p
fix-app = λ f → cong-app (fix-fix f)

{-# REWRITE fix-app 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

```

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

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

```

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 ( $\lambda g. \lambda a. \text{ifz } a \text{ then } 42 \text{ else } g (\text{pred } a)$ ) 101  $\equiv 42$ 
check-countdown :
   $\mathcal{A}' \llbracket Y \sqcup (\bar{\lambda} g \sqcup \bar{\lambda} a \sqcup$ 
     $(\text{if} \sqcup (Z \sqcup V a) \sqcup N 42 \sqcup (V g \sqcup (\text{pred} \sqcup V a))))$ 
     $\sqcup N 101$ 
   $\rrbracket \rho \perp$ 
   $\equiv \eta 42$ 
check-countdown = refl

-- fix ( $\lambda h. \lambda a. \lambda b. \text{ifz } a \text{ then } b \text{ else } h (\text{pred } a) (\text{succ } b)$ ) 4 38  $\equiv 42$ 
check-sum-42 :
   $\mathcal{A}' \llbracket (Y \sqcup (\bar{\lambda} h \sqcup \bar{\lambda} a \sqcup \bar{\lambda} b \sqcup$ 
     $(\text{if} \sqcup (Z \sqcup V a) \sqcup V b \sqcup (V h \sqcup (\text{pred} \sqcup V a) \sqcup (\text{succ} \sqcup V b))))$ 
     $\sqcup N 4 \sqcup N 38$ 
   $\rrbracket \rho \perp$ 
   $\equiv \eta 42$ 
check-sum-42 = refl
-- Exponential in first arg?

```

2 PCF.Constants

```

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 ( $\eta$ ;  $\_SHARP$ ; fix;  $\perp$ ;  $\_ \longrightarrow \_, \_$ )
open import PCF.Types
using (Types; o;  $\iota$ ;  $\_ \Rightarrow \_$ ;  $\sigma$ ;  $\mathcal{D}$ )

-- Syntax

data  $\mathcal{L}$  : Types  $\rightarrow$  Set where
  tt   :  $\mathcal{L}$  o
  ff   :  $\mathcal{L}$  o
   $\supset_i$  :  $\mathcal{L}$  (o  $\Rightarrow \iota \Rightarrow \iota \Rightarrow \iota$ )
   $\supset_o$  :  $\mathcal{L}$  (o  $\Rightarrow$  o  $\Rightarrow$  o  $\Rightarrow$  o)
  Y    : { $\sigma$  : Types}  $\rightarrow$   $\mathcal{L}$  (( $\sigma \Rightarrow \sigma$ )  $\Rightarrow \sigma$ )
  k    : (n : Nat)  $\rightarrow$   $\mathcal{L}$   $\iota$ 
  +1'  :  $\mathcal{L}$  ( $\iota \Rightarrow \iota$ )
  -1'  :  $\mathcal{L}$  ( $\iota \Rightarrow \iota$ )
  Z    :  $\mathcal{L}$  ( $\iota \Rightarrow$  o)

variable c :  $\mathcal{L}$   $\sigma$ 

-- Semantics

 $\mathcal{A}[\_]$  :  $\mathcal{L}$   $\sigma \rightarrow \mathcal{D}$   $\sigma$ 

 $\mathcal{A}[\text{tt}] = \eta \text{ true}$ 
 $\mathcal{A}[\text{ff}] = \eta \text{ false}$ 
 $\mathcal{A}[\supset_i] = \_ \longrightarrow \_, \_$ 
 $\mathcal{A}[\supset_o] = \_ \longrightarrow \_, \_$ 
 $\mathcal{A}[Y] = \text{fix}$ 
 $\mathcal{A}[k\ n] = \eta\ n$ 
 $\mathcal{A}[+1'] = (\lambda\ n \rightarrow \eta\ (n + 1))\ SHARP$ 
 $\mathcal{A}[-1'] = (\lambda\ n \rightarrow \text{if } n == 0 \text{ then } \perp \text{ else } \eta\ (n - 1))\ SHARP$ 
 $\mathcal{A}[Z] = (\lambda\ n \rightarrow \eta\ (n == 0))\ SHARP$ 

```

3 PCF.Domain-Notation

```

module PCF.Domain-Notation where

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

variable D E : Set -- Set should be a sort of domains

-- Domains are pointed
postulate
  ⊥ : {D : Set} → D

-- Fixed points of endofunctions on function domains

postulate
  fix : {D : Set} → (D → D) → D

  -- Properties
  fix-fix : ∀ {D} (f : D → D) → fix f ≡ f (fix f)

-- Lifted domains

postulate
  ℒ      : Set → Set
  η      : {P : Set} → P → ℒ P
  _ SHARP : {P : Set} {D : Set} → (P → D) → (ℒ P → D)

  -- Properties
  elim-SHARP-η : ∀ {P D} (f : P → D) (p : P) → (f SHARP) (η p) ≡ f p
  elim-SHARP-⊥ : ∀ {P D} (f : P → D) → (f SHARP) ⊥ ≡ ⊥

-- Flat domains

_+⊥      : Set → Set
S+⊥      = ℒ S

-- 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
  _ ⟶ _ , _ : {D : Set} → Bool +⊥ → D → D → D

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

```

4 PCF.Environments

```

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

```

module PCF.Terms where

open import PCF.Types
  using (Types;  $\_ \Rightarrow \_$ ;  $\sigma$ ;  $\mathcal{D}$ )
open import PCF.Constants
  using ( $\mathcal{L}$ ;  $\mathcal{A}[\_]$ ;  $c$ )
open import PCF.Variables
  using ( $\mathcal{V}$ ; Env;  $\_ \llbracket \_ \rrbracket$ )
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  $\rightarrow$   $\mathcal{D} \sigma$ 

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

```


6 PCF.Types

```
module PCF.Types where

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

open import PCF.Domain-Notation
using ( _ +⊥ )

-- Syntax

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

variable σ τ : Types

infixr 1 _ ⇒ _

-- Semantics  $\mathcal{D}$ 

 $\mathcal{D}$  : Types → Set -- Set should be a sort of domains

 $\mathcal{D} \ell$       = Nat +⊥
 $\mathcal{D} o$        = Bool +⊥
 $\mathcal{D} (\sigma \Rightarrow \tau)$  =  $\mathcal{D} \sigma \rightarrow \mathcal{D} \tau$ 

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

7 PCF.Variables

```
module PCF.Variables where

open import Agda.Builtin.Nat
using (Nat)

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 =  $\forall \{\sigma\} \rightarrow \mathcal{V}$   $\sigma \rightarrow \mathcal{D}$   $\sigma$ 

variable  $\rho$  : Env

-- Semantics

_  $\llbracket$  _  $\rrbracket$  : Env  $\rightarrow$   $\mathcal{V}$   $\sigma \rightarrow \mathcal{D}$   $\sigma$ 

 $\rho$   $\llbracket$   $\alpha$   $\rrbracket$  =  $\rho$   $\alpha$ 
```

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
```