

# PCF.index

November 20, 2025

## Contents

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

# 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 ℓ
g = var 1 (ℓ ⇒ ℓ)
h = var 2 (ℓ ⇒ ℓ ⇒ ℓ)
a = var 3 ℓ
b = var 4 ℓ

-- Arithmetic
check-41+1 : A'[ succ ⊑ N 41 ] ρ⊥ ≡ η 42
check-41+1 = refl

check-43-1 : A'[ pred⊥ ⊑ N 43 ] ρ⊥ ≡ η 42
check-43-1 = refl

-- Binding
check-id : A'[ (λ a ⊑ V a) ⊑ N 42 ] ρ⊥ ≡ η 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{\lambda} a \u (succ \u V a)) \u N 41] \rho\perp \equiv \eta 42
check-suc-41 = refl

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

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

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

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

-- fix (\lambda g. \lambda a. ifz a then 42 else g (pred a)) 101 \equiv 42
check-countdown :
  A'[Y \u (\bar{\lambda} g \u \bar{\lambda} a \u
    (if \u (Z \u V a) \u N 42 \u (V g \u (pred\perp \u V a)))) \u N 101]
  \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)) 4 38 \equiv 42
check-sum-42 :
  A'[(Y \u (\bar{\lambda} h \u \bar{\lambda} a \u \bar{\lambda} b \u
    (if \u (Z \u V a) \u V b \u (V h \u (pred\perp \u V a) \u (succ \u V b)))) \u N 4 \u N 38)
  ] \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 (η; _SHARP; fix; ⊥; _→_ , _)
open import PCF.Types
  using (Types; o; ℓ; _⇒_ ; σ; ℐ)

-- 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 → ℐ 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

```

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

S_+⊥      : 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; ↩; ⚡; _ ⇒ _ ; ℰ)
open import PCF.Variables
  using (𝕍; var; Env)

-- ρ⊥ is the initial environment

ρ⊥ : Env
ρ⊥ ⚡ = ⊥

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

_ [ _ / _ ] : {σ : Types} → Env → ℰ σ → ∨ σ → Env
ρ [ x / α ] = λ α' → h ρ × α α' (α ≡∨ α') where
  h : {σ τ : Types} → Env → ℰ σ → ∨ σ → ∨ τ → Maybe (σ ≡ τ) → ℰ 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 }

_ ==T ↩ = just refl
_ ==T ⚡ = just refl
_ ==T _ = nothing

_ ==V_ : {σ τ : Types} → ∨ σ → ∨ 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 (_⇒_; σ; ℐ)
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 → ℐ 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

```
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 → Types -- functions

variable σ τ : Types

infixr 1 _⇒_

-- Semantics D

D : Types → Set -- Set should be a sort of domains

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

variable x y z : D σ
```

## 7 PCF.Variables

```
module PCF.Variables where

open import Agda.Builtin.Nat
using (Nat)

open import PCF.Types
using (Types; σ; ℰ)

-- Syntax

data V : Types → Set where
  var : Nat → (σ : Types) → V σ

variable α : V σ

-- Environments

Env = ∀ {σ} → V σ → ℰ σ

variable ρ : Env

-- Semantics

_⟦_⟧ : Env → V σ → ℰ 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
```