

LC.index

November 29, 2025

Contents

1	LC.Definitions	2
2	LC.Domains	3
3	LC.Environments	4
4	LC.Semantics	5
5	LC.Terms	6
6	LC.Tests	7
7	LC.Variables	9
8	LC.index	10

1 LC.Definitions

```
module LC.Definitions where

import LC.Variables
import LC.Terms
import LC.Domains
import LC.Environments
import LC.Semantics
```

2 LC.Domains

```
module LC.Domains where

open import Function
  using (Inverse; _ ↔ _ ) public
open Inverse {{ ... }}
  using (to; from) public

postulate
  Domain : Set1
  ⟨⟨ _ ⟩⟩ : Domain → Set
  D∞ : Domain
postulate
  instance iso : ⟨⟨ D∞ ⟩⟩ ↔ (⟨⟨ D∞ ⟩⟩ → ⟨⟨ D∞ ⟩⟩)

variable d : ⟨⟨ D∞ ⟩⟩
```

The PCF example illustrates declaration of a domain of functions.

3 LC.Environments

```
module LC.Environments where

open import LC.Variables
open import LC.Domains
open import Data.Bool using (if _ then _ else _)

Env = Var → ⟨⟨ D∞ ⟩⟩

variable ρ : Env

_ [ _ / _ ] : Env → ⟨⟨ D∞ ⟩⟩ → Var → Env
ρ [ d / v ] = λ v' → if v == v' then d else ρ v'
```

4 LC.Semantics

```
module LC.Semantics where

open import LC.Variables
open import LC.Terms
open import LC.Domains
open import LC.Environments

[[ _ ]] : Exp → Env → ⟨⟨ D∞ ⟩⟩
-- [[ e ]] ρ is the value of e with ρ giving the values of free variables

[[ var v      ]] ρ = ρ v
[[ lam v e    ]] ρ = from ( λ d → [[ e ]] (ρ [ d / v ]) )
[[ app e1 e2 ]] ρ = to ( [[ e1 ]] ρ ) ( [[ e2 ]] ρ )
```

5 LC.Terms

```
module LC.Terms where

open import LC.Variables

data Exp : Set where
  var _ : Var → Exp          -- variable value
  lam  : Var → Exp → Exp    -- lambda abstraction
  app  : Exp → Exp → Exp    -- application

variable e : Exp
```

6 LC.Tests

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

open import Agda.Builtin.Equality
open import Agda.Builtin.Equality.Rewrite

module LC.Tests where

open import LC.Domains
open import LC.Variables
open import LC.Terms
open import LC.Semantics

open import Relation.Binary.PropositionalEquality using (refl)
open Inverse using (inversel; inverser)

to-from-elim : ∀ {f} → to (from f) ≡ f
to-from-elim = inversel iso refl

from-to-elim : ∀ {d} → from (to d) ≡ d
from-to-elim = inverser iso refl

{-# REWRITE to-from-elim #-}

-- The following proofs are potentially unsound,
-- due to rewriting using the postulated iso

-- (λx1.x1)x42 = x42
check-id :
  [ app (lam (x 1) (var x 1))
    (var x 42) ] ≡ [ var x 42 ]
check-id = refl

-- (λx1.x42)x0 = x42
check-const :
  [ app (lam (x 1) (var x 42))
    (var x 0) ] ≡ [ var x 42 ]
check-const = refl

-- (λx0.x0 x0)(λx0.x0 x0) = ...
-- check-divergence :
--   [ app (lam (x 0) (app (var x 0) (var x 0)))
--     (lam (x 0) (app (var x 0) (var x 0))) ]
--   ≡ [ var x 42 ]
-- check-divergence = refl

-- (λx1.x42)((λx0.x0 x0)(λx0.x0 x0)) = x42
check-convergence :
  [ app (lam (x 1) (var x 42))
    (app (lam (x 0) (app (var x 0) (var x 0)))
      (lam (x 0) (app (var x 0) (var x 0)))) ]
```

```

≡ [ [ var x 42 ] ]
check-convergence = refl

-- (λx1.x1)(λx1.x42) = λx2.x42
check-abs :
  [ [ app (lam (x 1) (var x 1))
        (lam (x 1) (var x 42)) ] ]
    ≡ [ [ lam (x 2) (var x 42) ] ]
check-abs = refl

-- (λx1.(λx42.x1)x2)x42 = x42
check-free :
  [ [ app (lam (x 1)
              (app (lam (x 42) (var x 1))
                    (var x 2)))
        (var x 42) ] ] ≡ [ [ var x 42 ] ]
check-free = refl

```


7 LC.Variables

```
module LC.Variables where

open import Data.Bool using (Bool)
open import Data.Nat using (ℕ;  $\equiv^b$  _)

data Var : Set where
  x : ℕ → Var -- variables

variable v : Var

_ == _ : Var → Var → Bool
x n == x n' = (n  $\equiv^b$  n')
```

8 LC.index

```
{-# OPTIONS --rewriting --confluence-check #-}  
  
module LC.index where  
  
import LC.Definitions  
import LC.Domains  
import LC.Environments  
import LC.Semantics  
import LC.Terms  
import LC.Tests  
import LC.Variables
```