Denotational Semantics of Untyped λ-Calculus in Agda

DRAFT (March 4, 2025)

Peter D. Mosses

Delft University of Technology, The Netherlands
p.d.mosses@tudelft.nl
Swansea University, United Kingdom

Abstract

In synthetic domain theory, all sets are predomains, domains are pointed sets, and functions are implicitly continuous. The denotational semantics of the untyped lambda-calculus presented here illustrates how it might look if synthetic domain theory can be implemented in Agda. Currently, the code uses unsatisfiable postulates as a work-around, to allow Agda to type-check the definitions.

The Agda source code used to generate this document is currently available only in a private repository, but will soon be made public.

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

module ULC.All where

import ULC.Variables
import ULC.Terms
import ULC.Domains
import ULC.Environments
import ULC.Semantics
import ULC.Checks
```

```
module ULC.Variables where

open import Data.Bool using (Bool)
open import Data.Nat using (ℕ; _≡ᵇ_)

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

variable v : Var

_==_ : Var → Var → Bool
x n == x n′ = (n ≡ᵇ n′)
```

```
module ULC.Terms where

open import ULC.Variables

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

variable e : Exp
```

```
module ULC.Domains where

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

Domain = Set

postulate ⊥       : {D : Domain} → D

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

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

postulate fix-app : ∀ {P D} → (f : (P → D) → (P → D)) (p : P) → fix f p ≡ f (fix f) p

open import Function using (Inverse; _↔_) public

postulate D∞ : Domain
postulate instance iso : D∞ ↔ (D∞ → D∞)
open Inverse {{ ... }} using (to; from) public

variable d : D∞
```

```
module ULC.Environments where

open import ULC.Variables
open import ULC.Domains
open import Data.Bool using (if_then_else_)

Env : Domain
Env = Var → D∞
-- the initial environment for a closed term is λ v → ⊥

variable ρ : Env

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

```
module ULC.Semantics where

open import ULC.Variables
open import ULC.Terms
open import ULC.Domains
open import ULC.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 )
```

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

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

module ULC.Checks where

open import ULC.Domains
open import ULC.Variables
open import ULC.Terms
open import ULC.Environments
open import ULC.Semantics

open Inverse using (inverseˡ; inverseʳ)

to-from : (f : D∞ → D∞) → to (from f) ≡ f
from-to : (d : D∞)       → from (to d) ≡ d

to-from f = inverseˡ iso refl
from-to f = inverseʳ iso refl

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

-- The following proofs are potentially unsound, due to unsafe postulates.

-- (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
```