Scm.index

December 23, 2025

Contents

1 Scm.Abstract-Syntax

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

module Scm.Abstract-Syntax where

open import Data.Integer.Base renaming (ℤ to Int) public
open import Data.String.Base  using (String) public

data     Con   : Set    -- constants, *excluding* quotations
variable K     : Con
Ide            = String -- identifiers (variables)
variable I     : Ide
data     Exp   : Set    -- expressions
variable E     : Exp
data     Exp   : Set    -- expression sequences
variable E     : Exp

data     Body  : Set    -- body expression or definition
variable B     : Body
data     Body⁺ : Set    -- body sequences
variable B⁺    : Body⁺
data     Prog  : Set    -- programs
variable Π     : Prog

------------------------------------------------------------------------
-- Literal constants

data Con where   -- basic constants
  int : Int → Con -- integer numerals
  #t : Con        -- true
  #f : Con        -- false

------------------------------------------------------------------------
-- Expressions

data Exp where                                  -- expressions
  con              : Con → Exp                  -- K
  ide              : Ide → Exp                  -- I
  ⦅ _ ␣ _ ⦆         : Exp → Exp → Exp            -- (E E)
  ⦅lambda _ ␣ _ ⦆   : Ide → Exp → Exp            -- (lambda I E)
  ⦅if _ ␣ _ ␣ _ ⦆   : Exp → Exp → Exp → Exp      -- (if E E₁ E₂)
  ⦅set! _ ␣ _ ⦆     : Ide → Exp → Exp            -- (set! I E)

data Exp where                                  -- expression sequences
  ␣␣␣              : Exp                        -- empty sequence
  _ ␣␣ _           : Exp → Exp → Exp            -- prefix sequence E E
```

```
--------------------------------------------------------------------
-- Definitions and Programs

data Body where
  ␣␣_              : Exp → Body                -- side-effect expression E
  (|define _ ␣ _ |) : Ide → Exp → Body         -- definition (define I E)
  (|begin _ |)     : Body⁺ → Body              -- block (begin B⁺)

data Body⁺ where                               -- body sequence
  ␣␣_              : Body → Body⁺              -- single body sequence B
  _ ␣␣ _           : Body → Body⁺ → Body⁺      -- prefix body sequence B B⁺

data Prog where                                -- programs
  ␣␣␣              : Prog                      -- empty program
  ␣␣_              : Body⁺ → Prog              -- non-empty program B⁺

infix 30 ␣␣_
infixr 20 _␣␣_
```

2 Scm.Auxiliary-Functions

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

module Scm.Auxiliary-Functions where

open import Scm.Notation
open import Scm.Abstract-Syntax
open import Scm.Domain-Equations

-- Environments ρ : U = Ide →ˢ L

postulate _==_ : Ide → Ide → Bool

_[_/_] : ⟪ U →ᶜ L →ᶜ Ide →ˢ U ⟫
ρ [ α / I ] = λ I′ → η (I == I′) ⟶ α , ρ I′

postulate unknown : ⟪ L ⟫
-- ρ I = unknown represents the lack of a binding for I in ρ

postulate initial-env : ⟪ U ⟫
-- initial-env shoud include various procedures and values

-- Stores σ : S = L →ᶜ E

_[_/_]′ : ⟪ S →ᶜ E →ᶜ L →ᶜ S ⟫
σ [ ϵ / α ]′ = λ α′ → (α ==ᴸ α′) ⟶ ϵ , σ α′

assign : ⟪ L →ᶜ E →ᶜ C →ᶜ C ⟫
assign = λ α ϵ θ σ → θ (σ [ ϵ / α ]′)

hold : ⟪ L →ᶜ (E →ᶜ C) →ᶜ C ⟫
hold = λ α κ σ → κ (σ α) σ

postulate new : ⟪ (L →ᶜ C) →ᶜ C ⟫
-- new κ σ = κ α σ′ where σ α = unallocated, σ′ α ≠ unallocated

alloc : ⟪ E →ᶜ (L →ᶜ C) →ᶜ C ⟫
alloc = λ ϵ κ → new (λ α → assign α ϵ (κ α))
-- should be ⊥ when ϵ |-M == unallocated

initial-store : ⟪ S ⟫
initial-store = λ α → η unallocated M-in-E

postulate finished : ⟪ C ⟫
-- normal termination with answer depending on final store

truish : ⟪ E →ᶜ T ⟫
truish =
  λ ϵ → (ϵ ∈-T) ⟶
      (((ϵ |-T) ==ᵀ η false) ⟶ η false , η true) ,
    η true
```

```
-- Lists

cons : ⟪ F ⟫
cons =
  λ ϵ κ →
      (# ϵ ==⊥ 2) ⟶ alloc (ϵ ↓ 1) (λ α₁ →
                      alloc (ϵ ↓ 2) (λ α₂ →
                        κ ((α₁ , α₂) -in-E))) ,
    ⊥

list : ⟪ F ⟫
list = fix {D = F} λ list′ →
  λ ϵ κ →
    (# ϵ ==⊥ 0) ⟶ κ (η null M-in-E) ,
      list′ (ϵ † 1) (λ ϵ → cons ⟨ (ϵ ↓ 1) , ϵ ⟩ κ)

car : ⟪ F ⟫
car =
  λ ϵ κ → (# ϵ ==⊥ 1) ⟶ hold ((ϵ ↓ 1) |- ↓²1) κ , ⊥

cdr : ⟪ F ⟫
cdr =
  λ ϵ κ → (# ϵ ==⊥ 1) ⟶ hold ((ϵ ↓ 1) |- ↓²2) κ , ⊥

setcar : ⟪ F ⟫
setcar =
  λ ϵ κ →
      (# ϵ ==⊥ 2) ⟶ assign ((ϵ ↓ 1) |- ↓²1)
                        (ϵ ↓ 2)
                        (κ (η unspecified M-in-E)) ,
    ⊥

setcdr : ⟪ F ⟫
setcdr =
  λ ϵ κ →
      (# ϵ ==⊥ 2) ⟶ assign ((ϵ ↓ 1) |- ↓²2)
                        (ϵ ↓ 2)
                        (κ (η unspecified M-in-E)) ,
    ⊥
```

3 Scm.Domain-Equations

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

module Scm.Domain-Equations where

open import Scm.Notation
open import Scm.Abstract-Syntax using (Ide; Int)

-- Domain declarations

postulate L  :  Domain -- locations
variable  α  :  ⟪ L ⟫
N            :  Domain -- natural numbers
T            :  Domain -- booleans
R            :  Domain -- numbers
             :  Domain -- pairs
M            :  Domain -- miscellaneous
variable  μ  :  ⟪ M ⟫
F            :  Domain -- procedure values
variable  φ  :  ⟪ F ⟫
postulate E  :  Domain -- expressed values
variable  ϵ  :  ⟪ E ⟫
S            :  Domain -- stores
variable  σ  :  ⟪ S ⟫
U            :  Domain -- environments
variable  ρ  :  ⟪ U ⟫
C            :  Domain -- command continuations
variable  θ  :  ⟪ C ⟫
postulate A  :  Domain -- answers

E            = E
variable  ϵ  :  ⟪ E ⟫

-- Domain equations

data Misc : Set where
  null unallocated undefined unspecified : Misc

N  = Nat⊥
T  = Bool⊥
R  = Int +⊥
   = L × L
M  = Misc +⊥
F  = E →ᶜ (E →ᶜ C) →ᶜ C
-- E  =  T + R +  + M + F
S  = L →ᶜ E
U  = Ide →ˢ L
C  = S →ᶜ A
```

```
-- Injections, tests, and projections

postulate
  _T-in-E : ⟪ T →ᶜ E ⟫
  _∈-T    : ⟪ E →ᶜ Bool +⊥ ⟫
  _|-T    : ⟪ E →ᶜ T ⟫

  _R-in-E : ⟪ R →ᶜ E ⟫
  _∈-R    : ⟪ E →ᶜ Bool +⊥ ⟫
  _|-R    : ⟪ E →ᶜ R ⟫

  _-in-E  : ⟪  →ᶜ E ⟫
  _∈-     : ⟪ E →ᶜ Bool +⊥ ⟫
  _|-     : ⟪ E →ᶜ  ⟫

  _M-in-E : ⟪ M →ᶜ E ⟫
  _∈-M    : ⟪ E →ᶜ Bool +⊥ ⟫
  _|-M    : ⟪ E →ᶜ M ⟫

  _F-in-E : ⟪ F →ᶜ E ⟫
  _∈-F    : ⟪ E →ᶜ Bool +⊥ ⟫
  _|-F    : ⟪ E →ᶜ F ⟫

-- Operations on flat domains

postulate
  _==ᴸ_  : ⟪ L →ᶜ L →ᶜ T ⟫
  _==ᴹ_  : ⟪ M →ᶜ M →ᶜ T ⟫
  _==ᴿ_  : ⟪ R →ᶜ R →ᶜ T ⟫
  _==ᵀ_  : ⟪ T →ᶜ T →ᶜ T ⟫
  _<ᴿ_   : ⟪ R →ᶜ R →ᶜ T ⟫
  _+ᴿ_   : ⟪ R →ᶜ R →ᶜ R ⟫
  _∧ᵀ_   : ⟪ T →ᶜ T →ᶜ T ⟫
```

4 Scm.Notation

```
{-# OPTIONS --rewriting --confluence-check --lossy-unification #-}
open import Agda.Builtin.Equality
open import Agda.Builtin.Equality.Rewrite

module Scm.Notation where

open import Data.Bool.Base   using (Bool; false; true) public
open import Data.Nat.Base    renaming (ℕ to Nat) using (suc) public
open import Data.String.Base using (String) public
open import Data.Unit.Base   using (⊤)
open import Function         using (id; _∘_) public

postulate
  Domain : Set₁
  ⟪_⟫    : Domain → Set

variable
  A B C  : Set
  D E F  : Domain
  n      : Nat

------------------------------------------------------------------------
-- Domains

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 : ⟪ A →ˢ E ⟫ ≡ (A → ⟪ E ⟫)

{-# REWRITE dom-cts set-cts #-}

postulate
  fix : ⟪ (D →ᶜ D) →ᶜ D ⟫ -- fixed point of endofunction

------------------------------------------------------------------------
-- Flat domains

postulate
  _+⊥      : Set → Domain                     -- lifted set
  η        : ⟪ A →ˢ A +⊥ ⟫                    -- inclusion
  _SHARP   : ⟪ (A →ˢ D) →ᶜ A +⊥ →ᶜ D ⟫        -- Kleisli extension

Bool⊥      = Bool +⊥                          -- truth value domain
Nat⊥       = Nat +⊥                           -- natural number domain
```

```
String⊥        = String +⊥                  -- meta-string domain

postulate
  _==⊥_   : ⟪ Nat⊥ →ᶜ Nat →ˢ Bool⊥ ⟫    -- strict numerical equality
  _>=⊥_   : ⟪ Nat⊥ →ᶜ Nat →ˢ Bool⊥ ⟫    -- strict greater or equal
  _⟶_,_   : ⟪ Bool⊥ →ᶜ D →ᶜ D →ᶜ D ⟫    -- McCarthy conditional

--------------------------------------------------------------------------
-- Sum domains

postulate
  _+_      : Domain → Domain → Domain -- separated sum
  inj₁     : ⟪ D →ᶜ D + E ⟫                          -- injection
  inj₂     : ⟪ E →ᶜ D + E ⟫                          -- injection
  [_,_]    : ⟪ (D →ᶜ F) →ᶜ (E →ᶜ F) →ᶜ (D + E →ᶜ F) ⟫ -- case analysis

--------------------------------------------------------------------------
-- Product domains

postulate
  _×_  : Domain → Domain → Domain -- cartesian product
  _,_  : ⟪ D →ᶜ E →ᶜ D × E ⟫          -- pairing
  _↓²1 : ⟪ D × E →ᶜ D ⟫              -- 1st projection
  _↓²2 : ⟪ D × E →ᶜ E ⟫              -- 2nd projection
  _↓³1 : ⟪ D × E × F →ᶜ D ⟫          -- 1st projection
  _↓³2 : ⟪ D × E × F →ᶜ E ⟫          -- 2nd projection
  _↓³3 : ⟪ D × E × F →ᶜ F ⟫          -- 3rd projection

--------------------------------------------------------------------------
-- Tuple domains

_^_ : Domain → Nat → Domain      -- D ^ n  n-tuples
D ^ 0           = ⊤ +⊥
D ^ 1           = D
D ^ suc (suc n) = D × (D ^ suc n)

--------------------------------------------------------------------------
-- Finite sequence domains

postulate
  _        : Domain → Domain      -- D  domain of finite sequences
  ⟨⟩       : ⟪ D  ⟫                -- empty sequence
  ⟨_⟩      : ⟪ (D ^ suc n) →ᶜ D  ⟫ -- ⟨ d₁ , ... , dₙ₊₁ ⟩ non-empty sequence
  #        : ⟪ D  →ᶜ Nat⊥ ⟫        -- # d             sequence length
  _§_      : ⟪ D  →ᶜ D  →ᶜ D  ⟫    -- d § d          concatenation
  _↓_      : ⟪ D  →ᶜ Nat →ˢ D ⟫    -- d ↓ n           nth component
  _†_      : ⟪ D  →ᶜ Nat →ˢ D  ⟫   -- d † n           nth tail

--------------------------------------------------------------------------
-- Grouping precedence

infixr 0   _→ᶜ_  infixr 0  _→ˢ_  infixr 1  _+_
```

```
infixr 2    _×_
infixr 4    _,_
infix 8     _^_
infix 10 _+⊥
infixr 20   _⟶_,_

-- ⟦_⟧ = id
```

5 Scm.Semantic-Functions

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

module Scm.Semantic-Functions where

open import Scm.Notation
open import Scm.Abstract-Syntax
open import Scm.Domain-Equations
open import Scm.Auxiliary-Functions

𝒦⟦ _ ⟧   : ⟪ Con →ˢ E ⟫
ℰ⟦ _ ⟧   : ⟪ Exp →ˢ U →ᶜ (E →ᶜ C) →ᶜ C ⟫
ℰ⟦ _ ⟧ : ⟪ Exp →ˢ U →ᶜ (E →ᶜ C) →ᶜ C ⟫

ℬ⟦ _ ⟧   : ⟪ Body →ˢ U →ᶜ (U →ᶜ C) →ᶜ C ⟫
ℬ⁺⟦ _ ⟧ : ⟪ Body⁺ →ˢ U →ᶜ (U →ᶜ C) →ᶜ C ⟫
𝒫⟦ _ ⟧   : ⟪ Prog →ˢ A ⟫

-- Constant denotations 𝒦⟦ K ⟧ : E

𝒦⟦ int Z ⟧  = η Z R-in-E
𝒦⟦ #t ⟧     = η true T-in-E
𝒦⟦ #f ⟧     = η false T-in-E

-- Expression denotations

ℰ⟦ con K ⟧ ρ κ = κ (𝒦⟦ K ⟧)

ℰ⟦ ide I ⟧ ρ κ = hold (ρ I) κ

ℰ⟦ ⦅ E ␣ E ⦆ ⟧ ρ κ =
  ℰ⟦ E ⟧ ρ (λ ϵ →
    ℰ⟦ E ⟧ ρ (λ ϵ →
      (ϵ |-F) ϵ κ))

ℰ⟦ ⦅lambda I ␣ E ⦆ ⟧ ρ κ =
  κ (  (λ ϵ κ′ →
          list ϵ (λ ϵ →
            alloc ϵ (λ α →
              ℰ⟦ E ⟧ (ρ [ α / I ]) κ′))
       ) F-in-E)

ℰ⟦ ⦅if E ␣ E₁ ␣ E₂ ⦆ ⟧ ρ κ =
  ℰ⟦ E ⟧ ρ (λ ϵ →
    truish ϵ ⟶ ℰ⟦ E₁ ⟧ ρ κ , ℰ⟦ E₂ ⟧ ρ κ)

ℰ⟦ ⦅set! I ␣ E ⦆ ⟧ ρ κ =
  ℰ⟦ E ⟧ ρ (λ ϵ →
    assign (ρ I) ϵ (
      κ (η unspecified M-in-E)))

-- ℰ⟦_⟧  : Exp → U → (E → C) → C
```

$$\mathcal{E}[\![\ \sqcup\sqcup\sqcup\ ]\!]\ \rho\ \kappa = \kappa\ \langle\rangle$$

$$\begin{aligned}
&\mathcal{E}[\![\ \mathsf{E}\ \sqcup\sqcup\ \mathsf{E}\ ]\!]\ \rho\ \kappa = \\
&\quad \mathcal{E}[\![\ \mathsf{E}\ ]\!]\ \rho\ (\lambda\ \epsilon \rightarrow \\
&\qquad \mathcal{E}[\![\ \mathsf{E}\ ]\!]\ \rho\ (\lambda\ \epsilon \rightarrow \\
&\qquad\quad \kappa\ (\langle\ \epsilon\ \rangle\ \S\ \epsilon)))
\end{aligned}$$

```
-- Body denotations 𝓑⟦ B ⟧ : U → (U → C) → C
```

$\mathcal{B}⟦ ␣␣ \mathsf{E} ⟧\ \rho\ \kappa = \mathcal{E}⟦ \mathsf{E} ⟧\ \rho\ (\lambda\ \epsilon \rightarrow \kappa\ \rho)$

$\mathcal{B}⟦ (\!|\mathsf{define\ I}\ ␣\ \mathsf{E}\ |\!)\ ⟧\ \rho\ \kappa =$
$\quad \mathcal{E}⟦ \mathsf{E} ⟧\ \rho\ (\lambda\ \epsilon \rightarrow (\rho\ \mathsf{I} ==^{L} \mathsf{unknown}) \longrightarrow$
$\qquad\qquad\qquad \mathsf{alloc}\ \epsilon\ (\lambda\ \alpha \rightarrow \kappa\ (\rho\ [\ \alpha\ /\ \mathsf{I}\ ])),$
$\qquad\qquad\quad \mathsf{assign}\ (\rho\ \mathsf{I})\ \epsilon\ (\kappa\ \rho))$

$\mathcal{B}⟦ (\!|\mathsf{begin}\ \mathsf{B}^{+}\ |\!)\ ⟧\ \rho\ \kappa = \mathcal{B}^{+}⟦ \mathsf{B}^{+} ⟧\ \rho\ \kappa$

```
-- Body sequence denotations 𝓑⁺⟦ B⁺ ⟧ : U → (U → C) → C
```

$\mathcal{B}^{+}⟦ ␣␣ \mathsf{B} ⟧\ \rho\ \kappa = \mathcal{B}⟦ \mathsf{B} ⟧\ \rho\ \kappa$

$\mathcal{B}^{+}⟦ \mathsf{B}\ ␣␣\ \mathsf{B}^{+} ⟧\ \rho\ \kappa = \mathcal{B}⟦ \mathsf{B} ⟧\ \rho\ (\lambda\ \rho' \rightarrow \mathcal{B}^{+}⟦ \mathsf{B}^{+} ⟧\ \rho'\ \kappa)$

```
-- Program denotations 𝓟⟦ Π ⟧ : A
```

$\mathcal{P}⟦ ␣␣␣ ⟧ = \mathsf{finished\ initial\text{-}store}$

$\mathcal{P}⟦ ␣␣\ \mathsf{B}^{+} ⟧ = \mathcal{B}^{+}⟦ \mathsf{B}^{+} ⟧\ \mathsf{initial\text{-}env}\ (\lambda\ \rho \rightarrow \mathsf{finished})\ \mathsf{initial\text{-}store}$

6 Scm.index

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

module Scm.index where

import Scm.Notation
import Scm.Abstract-Syntax
import Scm.Domain-Equations
import Scm.Semantic-Functions
import Scm.Auxiliary-Functions
```