

# Scm.index

November 29, 2025

## Contents

<b>1</b>	<b>Scm.Abstract-Syntax</b>	<b>2</b>
<b>2</b>	<b>Scm.Auxiliary-Functions</b>	<b>4</b>
<b>3</b>	<b>Scm.Domain-Equations</b>	<b>6</b>
<b>4</b>	<b>Scm.Notation</b>	<b>8</b>
<b>5</b>	<b>Scm.Semantic-Functions</b>	<b>11</b>
<b>6</b>	<b>Scm.index</b>	<b>14</b>

# 1 Scm.Abstract-Syntax

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

```
module Scm.Abstract-Syntax where
```

```
open import Data.Integer.Base renaming ( $\mathbb{Z}$  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  $\Pi$  : 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
   $\sqcup\sqcup\sqcup$  : Exp -- empty sequence
  _  $\sqcup\sqcup$  _ : Exp → Exp → Exp -- prefix sequence E E
```

```

-----
-- Definitions and Programs

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

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

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

infix 30 UU _
infixr 20 _ UU _

```

## 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  $\rho : U = \text{Ide} \rightarrow^s L$ 

postulate _==_ : Ide  $\rightarrow$  Ide  $\rightarrow$  Bool

_[_ / _] :  $\langle\langle U \rightarrow^c L \rightarrow^c \text{Ide} \rightarrow^s U \rangle\rangle$ 
 $\rho [\alpha / l] = \lambda l' \rightarrow \eta (l == l') \rightarrow \alpha, \rho l'$ 

postulate unknown :  $\langle\langle L \rangle\rangle$ 
--  $\rho l = \text{unknown}$  represents the lack of a binding for  $l$  in  $\rho$ 

postulate initial-env :  $\langle\langle U \rangle\rangle$ 
-- initial-env should include various procedures and values

-- Stores  $\sigma : S = L \rightarrow^c E$ 

_[_ / _]' :  $\langle\langle S \rightarrow^c E \rightarrow^c L \rightarrow^c S \rangle\rangle$ 
 $\sigma [\epsilon / \alpha]' = \lambda \alpha' \rightarrow (\alpha ==^L \alpha') \rightarrow \epsilon, \sigma \alpha'$ 

assign :  $\langle\langle L \rightarrow^c E \rightarrow^c C \rightarrow^c C \rangle\rangle$ 
assign =  $\lambda \alpha \epsilon \theta \sigma \rightarrow \theta (\sigma [\epsilon / \alpha]')$ 

hold :  $\langle\langle L \rightarrow^c (E \rightarrow^c C) \rightarrow^c C \rangle\rangle$ 
hold =  $\lambda \alpha \kappa \sigma \rightarrow \kappa (\sigma \alpha) \sigma$ 

postulate new :  $\langle\langle (L \rightarrow^c C) \rightarrow^c C \rangle\rangle$ 
-- new  $\kappa \sigma = \kappa \alpha \sigma'$  where  $\sigma \alpha = \text{unallocated}$ ,  $\sigma' \alpha \neq \text{unallocated}$ 

alloc :  $\langle\langle E \rightarrow^c (L \rightarrow^c C) \rightarrow^c C \rangle\rangle$ 
alloc =  $\lambda \epsilon \kappa \rightarrow \text{new } (\lambda \alpha \rightarrow \text{assign } \alpha \epsilon (\kappa \alpha))$ 
-- should be  $\perp$  when  $\epsilon \mid\text{-M} == \text{unallocated}$ 

initial-store :  $\langle\langle S \rangle\rangle$ 
initial-store =  $\lambda \alpha \rightarrow \eta \text{unallocated M-in-E}$ 

postulate finished :  $\langle\langle C \rangle\rangle$ 
-- normal termination with answer depending on final store

truish :  $\langle\langle E \rightarrow^c T \rangle\rangle$ 
truish =
   $\lambda \epsilon \rightarrow (\epsilon \in \text{-T}) \rightarrow$ 
     $((\epsilon \mid\text{-T}) ==^T \eta \text{false}) \rightarrow \eta \text{false}, \eta \text{true},$ 
     $\eta \text{true}$ 

```

```

-- 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 (T)
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      : ⟨⟨ 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

 $\mathcal{K}[\_] : \langle \langle \text{Con} \rightarrow^s \mathbf{E} \rangle \rangle$ 
 $\mathcal{E}[\_] : \langle \langle \text{Exp} \rightarrow^s \mathbf{U} \rightarrow^c (\mathbf{E} \rightarrow^c \mathbf{C}) \rightarrow^c \mathbf{C} \rangle \rangle$ 
 $\mathcal{E}[\_] : \langle \langle \text{Exp} \rightarrow^s \mathbf{U} \rightarrow^c (\mathbf{E} \rightarrow^c \mathbf{C}) \rightarrow^c \mathbf{C} \rangle \rangle$ 

 $\mathcal{B}[\_] : \langle \langle \text{Body} \rightarrow^s \mathbf{U} \rightarrow^c (\mathbf{U} \rightarrow^c \mathbf{C}) \rightarrow^c \mathbf{C} \rangle \rangle$ 
 $\mathcal{B}^+[\_] : \langle \langle \text{Body}^+ \rightarrow^s \mathbf{U} \rightarrow^c (\mathbf{U} \rightarrow^c \mathbf{C}) \rightarrow^c \mathbf{C} \rangle \rangle$ 
 $\mathcal{P}[\_] : \langle \langle \text{Prog} \rightarrow^s \mathbf{A} \rangle \rangle$ 

-- Constant denotations  $\mathcal{K}[\mathbf{K}] : \mathbf{E}$ 

 $\mathcal{K}[\text{int } \mathbf{Z}] = \eta \ \mathbf{Z} \ \mathbf{R-in-E}$ 
 $\mathcal{K}[\#t] = \eta \ \text{true} \ \mathbf{T-in-E}$ 
 $\mathcal{K}[\#f] = \eta \ \text{false} \ \mathbf{T-in-E}$ 

-- Expression denotations

 $\mathcal{E}[\text{con } \mathbf{K}] \ \rho \ \kappa = \kappa \ (\mathcal{K}[\mathbf{K}])$ 

 $\mathcal{E}[\text{ide } \mathbf{l}] \ \rho \ \kappa = \text{hold} \ (\rho \ \mathbf{l}) \ \kappa$ 

 $\mathcal{E}[(\mathbf{E} \sqcup \mathbf{E})] \ \rho \ \kappa =$ 
 $\mathcal{E}[\mathbf{E}] \ \rho \ (\lambda \epsilon \rightarrow$ 
 $\mathcal{E}[\mathbf{E}] \ \rho \ (\lambda \epsilon \rightarrow$ 
 $(\epsilon \vdash \mathbf{F}) \ \kappa))$ 

 $\mathcal{E}[(\text{lambda } \mathbf{l} \sqcup \mathbf{E})] \ \rho \ \kappa =$ 
 $\kappa \ ($ 
 $(\lambda \epsilon \kappa' \rightarrow$ 
 $\text{list } \epsilon \ (\lambda \epsilon \rightarrow$ 
 $\text{alloc } \epsilon \ (\lambda \alpha \rightarrow$ 
 $\mathcal{E}[\mathbf{E}] \ (\rho \ [\alpha / \mathbf{l}]) \ \kappa'))$ 
 $) \ \mathbf{F-in-E})$ 

 $\mathcal{E}[(\text{if } \mathbf{E} \sqcup \mathbf{E}_1 \sqcup \mathbf{E}_2)] \ \rho \ \kappa =$ 
 $\mathcal{E}[\mathbf{E}] \ \rho \ (\lambda \epsilon \rightarrow$ 
 $\text{truish } \epsilon \longrightarrow \mathcal{E}[\mathbf{E}_1] \ \rho \ \kappa, \mathcal{E}[\mathbf{E}_2] \ \rho \ \kappa)$ 

 $\mathcal{E}[(\text{set! } \mathbf{l} \sqcup \mathbf{E})] \ \rho \ \kappa =$ 
 $\mathcal{E}[\mathbf{E}] \ \rho \ (\lambda \epsilon \rightarrow$ 
 $\text{assign} \ (\rho \ \mathbf{l}) \ \epsilon \ ($ 
 $\kappa \ (\eta \ \text{unspecified } \mathbf{M-in-E}))$ 

--  $\mathcal{E}[\_] : \text{Exp} \rightarrow \mathbf{U} \rightarrow (\mathbf{E} \rightarrow \mathbf{C}) \rightarrow \mathbf{C}$ 

```

$$\begin{aligned}
&\mathcal{E}[\llbracket \text{uuu} \rrbracket] \rho \kappa = \kappa \langle \rangle \\
&\mathcal{E}[\llbracket \text{E uu E} \rrbracket] \rho \kappa = \\
&\quad \mathcal{E}[\llbracket \text{E} \rrbracket] \rho (\lambda \epsilon \rightarrow \\
&\quad \quad \mathcal{E}[\llbracket \text{E} \rrbracket] \rho (\lambda \epsilon \rightarrow \\
&\quad \quad \quad \kappa (\langle \epsilon \rangle \S \epsilon)))
\end{aligned}$$

```

-- Body denotations  $\mathcal{B} \llbracket B \rrbracket : U \rightarrow (U \rightarrow C) \rightarrow C$ 

 $\mathcal{B} \llbracket \sqcup E \rrbracket \rho \kappa = \mathcal{E} \llbracket E \rrbracket \rho (\lambda \epsilon \rightarrow \kappa \rho)$ 

 $\mathcal{B} \llbracket (\text{define } l \sqcup E) \rrbracket \rho \kappa =$ 
 $\mathcal{E} \llbracket E \rrbracket \rho (\lambda \epsilon \rightarrow (\rho \models^L \text{unknown}) \rightarrow$ 
 $\quad \text{alloc } \epsilon (\lambda \alpha \rightarrow \kappa (\rho [\alpha / l])),$ 
 $\quad \text{assign } (\rho l) \epsilon (\kappa \rho))$ 

 $\mathcal{B} \llbracket (\text{begin } B^+) \rrbracket \rho \kappa = \mathcal{B}^+ \llbracket B^+ \rrbracket \rho \kappa$ 

-- Body sequence denotations  $\mathcal{B}^+ \llbracket B^+ \rrbracket : U \rightarrow (U \rightarrow C) \rightarrow C$ 

 $\mathcal{B}^+ \llbracket \sqcup B \rrbracket \rho \kappa = \mathcal{B} \llbracket B \rrbracket \rho \kappa$ 

 $\mathcal{B}^+ \llbracket B \sqcup B^+ \rrbracket \rho \kappa = \mathcal{B} \llbracket B \rrbracket \rho (\lambda \rho' \rightarrow \mathcal{B}^+ \llbracket B^+ \rrbracket \rho' \kappa)$ 

-- Program denotations  $\mathcal{P} \llbracket \Pi \rrbracket : A$ 

 $\mathcal{P} \llbracket \sqcup \sqcup \sqcup \rrbracket = \text{finished initial-store}$ 

 $\mathcal{P} \llbracket \sqcup B^+ \rrbracket = \mathcal{B}^+ \llbracket B^+ \rrbracket \text{initial-env } (\lambda \rho \rightarrow \text{finished}) \text{initial-store}$ 

```

## 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
```