

Scm.index

November 29, 2025

Contents

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

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 P : Prog

-----
-- Literal constants

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

-----
-- Expressions

data Exp where
  con          : Con → Exp           -- expressions
  ide          : Ide → Exp           -- K
  (λ _ _ )     : Exp → Exp → Exp    -- I
  (λ λ _ _ _ ) : 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
  uuu       : Exp           -- expression sequences
  _ uu_ : Exp → Exp → Exp -- empty sequence
  _ uu _ : Exp → Exp → Exp -- prefix sequence E E
```

```

-----  

-- Definitions and Programs  

  

data Body where
  uu_ : Exp → Body          -- side-effect expression E
  (define _ uu_) : 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+ → Body+ -- prefix body sequence B B+  

  

data Prog where
  uuu : 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 → Ide → Bool
 $\underline{\rho}[\underline{\_}/\underline{\_}] : \langle\!\langle U \rightarrow^c L \rightarrow^c \text{Ide} \rightarrow^s U \rangle\!\rangle$ 
 $\rho[\alpha/\beta] = \lambda \beta' \rightarrow \eta(\beta == \beta') \longrightarrow \alpha, \rho \beta'$ 

postulate unknown : ⟨⟨ L ⟩⟩
--  $\rho I = \text{unknown}$  represents the lack of a binding for  $I$  in  $\rho$ 

postulate initial-env : ⟨⟨ U ⟩⟩
-- initial-env shoud include various procedures and values

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

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

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

hold : ⟨⟨ L →^c (E →^c C) →^c C ⟩⟩
hold =  $\lambda \alpha \kappa \sigma \rightarrow \kappa(\sigma \alpha) \sigma$ 

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

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

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

postulate finished : ⟨⟨ C ⟩⟩
-- normal termination with answer depending on final store

truish : ⟨⟨ E →^c T ⟩⟩
truish =
 $\lambda \epsilon \rightarrow (\epsilon \in \neg T) \longrightarrow$ 
 $((\epsilon \mid T) ==^T \eta \text{ false}) \longrightarrow \eta \text{ false}, \eta \text{ true} ,$ 
 $\eta \text{ true}$ 
```

```

-- Lists

cons : << F >>
cons =

$$\lambda \epsilon \kappa \rightarrow$$


$$(\# \epsilon ==\perp 2) \longrightarrow \text{alloc } (\epsilon \downarrow 1) (\lambda \alpha_1 \rightarrow$$


$$\text{alloc } (\epsilon \downarrow 2) (\lambda \alpha_2 \rightarrow$$


$$\kappa ((\alpha_1 , \alpha_2) \text{-in-}\mathbf{E})) ,$$


$$\perp$$


list : << F >>
list = fix {D = F}  $\lambda \text{list}' \rightarrow$ 

$$\lambda \epsilon \kappa \rightarrow$$


$$(\# \epsilon ==\perp 0) \longrightarrow \kappa (\eta \text{ null M-in-}\mathbf{E}) ,$$


$$\text{list}' (\epsilon \dagger 1) (\lambda \epsilon \rightarrow \text{cons } \langle (\epsilon \downarrow 1) , \epsilon \rangle \kappa)$$


car : << F >>
car =

$$\lambda \epsilon \kappa \rightarrow (\# \epsilon ==\perp 1) \longrightarrow \text{hold } ((\epsilon \downarrow 1) \dashv \downarrow^2 1) \kappa , \perp$$


cdr : << F >>
cdr =

$$\lambda \epsilon \kappa \rightarrow (\# \epsilon ==\perp 1) \longrightarrow \text{hold } ((\epsilon \downarrow 1) \dashv \downarrow^2 2) \kappa , \perp$$


setcar : << F >>
setcar =

$$\lambda \epsilon \kappa \rightarrow$$


$$(\# \epsilon ==\perp 2) \longrightarrow \text{assign } ((\epsilon \downarrow 1) \dashv \downarrow^2 1)$$


$$(\epsilon \downarrow 2)$$


$$(\kappa (\eta \text{ unspecified M-in-}\mathbf{E})) ,$$


$$\perp$$


setcdr : << F >>
setcdr =

$$\lambda \epsilon \kappa \rightarrow$$


$$(\# \epsilon ==\perp 2) \longrightarrow \text{assign } ((\epsilon \downarrow 1) \dashv \downarrow^2 2)$$


$$(\epsilon \downarrow 2)$$


$$(\kappa (\eta \text{ unspecified M-in-}\mathbf{E})) ,$$


$$\perp$$


```

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 (Id; Int)

-- Domain declarations

postulate L : Domain -- locations
variable α : ⟨⟨ L ⟩⟩
N : Domain -- natural numbers
T : Domain -- booleans
R : Domain -- numbers
P : 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 + P + M + F
S = L →c E
U = Id ; →s L
C = S →c A
```

```
-- Injections, tests, and projections
```

```
postulate
```

$$\begin{aligned} \underline{\text{T-in-E}} &: \langle\langle \mathbf{T} \rightarrow^c \mathbf{E} \rangle\rangle \\ \underline{\in-T} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{Bool} + \perp \rangle\rangle \\ \underline{|-T} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{T} \rangle\rangle \\ \\ \underline{\text{R-in-E}} &: \langle\langle \mathbf{R} \rightarrow^c \mathbf{E} \rangle\rangle \\ \underline{\in-R} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{Bool} + \perp \rangle\rangle \\ \underline{|-R} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{R} \rangle\rangle \\ \\ \underline{-in-E} &: \langle\langle _ \rightarrow^c \mathbf{E} \rangle\rangle \\ \underline{\in-} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{Bool} + \perp \rangle\rangle \\ \underline{|-} &: \langle\langle \mathbf{E} \rightarrow^c _ \rangle\rangle \\ \\ \underline{\text{M-in-E}} &: \langle\langle \mathbf{M} \rightarrow^c \mathbf{E} \rangle\rangle \\ \underline{\in-M} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{Bool} + \perp \rangle\rangle \\ \underline{|-M} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{M} \rangle\rangle \\ \\ \underline{\text{F-in-E}} &: \langle\langle \mathbf{F} \rightarrow^c \mathbf{E} \rangle\rangle \\ \underline{\in-F} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{Bool} + \perp \rangle\rangle \\ \underline{|-F} &: \langle\langle \mathbf{E} \rightarrow^c \mathbf{F} \rangle\rangle \end{aligned}$$

```
-- Operations on flat domains
```

```
postulate
```

$$\begin{aligned} \underline{==^L_} &: \langle\langle \mathbf{L} \rightarrow^c \mathbf{L} \rightarrow^c \mathbf{T} \rangle\rangle \\ \underline{==^M_} &: \langle\langle \mathbf{M} \rightarrow^c \mathbf{M} \rightarrow^c \mathbf{T} \rangle\rangle \\ \underline{==^R_} &: \langle\langle \mathbf{R} \rightarrow^c \mathbf{R} \rightarrow^c \mathbf{T} \rangle\rangle \\ \underline{==^T_} &: \langle\langle \mathbf{T} \rightarrow^c \mathbf{T} \rightarrow^c \mathbf{T} \rangle\rangle \\ \underline{<^R_} &: \langle\langle \mathbf{R} \rightarrow^c \mathbf{R} \rightarrow^c \mathbf{T} \rangle\rangle \\ \underline{+^R_} &: \langle\langle \mathbf{R} \rightarrow^c \mathbf{R} \rightarrow^c \mathbf{R} \rangle\rangle \\ \underline{\wedge^T_} &: \langle\langle \mathbf{T} \rightarrow^c \mathbf{T} \rightarrow^c \mathbf{T} \rangle\rangle \end{aligned}$$

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; _ o _) 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 = T +⊥
  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

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

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

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

-- Expression denotations
 $\mathcal{E}[\text{con } K]$   $\rho \kappa$  =  $\kappa(\mathcal{K}[K])$ 
 $\mathcal{E}[\text{ide } I]$   $\rho \kappa$  =  $\text{hold } (\rho I) \kappa$ 

$$\begin{aligned} \mathcal{E}[(\text{E} \sqcup \text{E})] \rho \kappa &= \\ \mathcal{E}[\text{E}] \rho (\lambda \epsilon \rightarrow & \\ \mathcal{E}[\text{E}] \rho (\lambda \epsilon \rightarrow & \\ (\epsilon \mid \mathbf{F}) \epsilon \kappa)) & \end{aligned}$$


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


$$\begin{aligned} \mathcal{E}[(\text{if } E \sqcup E_1 \sqcup E_2)] \rho \kappa &= \\ \mathcal{E}[\text{E}] \rho (\lambda \epsilon \rightarrow & \\ \text{truish } \epsilon \longrightarrow \mathcal{E}[\text{E}_1] \rho \kappa, \mathcal{E}[\text{E}_2] \rho \kappa) & \end{aligned}$$


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

--  $\mathcal{E}[\_]$  : Exp → U → (E → C) → C
```

$$\mathcal{E}[\![\textcolor{brown}{\text{uuu}}]\!] \rho \kappa = \kappa \langle \rangle$$

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

```

-- Body denotations  $\mathcal{B}[\![\ B \ ]\!]$  :  $\mathbf{U} \rightarrow (\mathbf{U} \rightarrow \mathbf{C}) \rightarrow \mathbf{C}$ 

$$\mathcal{B}[\![\ \sqcup\ E \ ]\!] \rho \kappa = \mathcal{E}[\![\ E \ ]\!] \rho (\lambda \epsilon \rightarrow \kappa \rho)$$


$$\begin{aligned} \mathcal{B}[\![\ (\text{define } l \sqcup E) \ ]\!] \rho \kappa = \\ \mathcal{E}[\![\ E \ ]\!] \rho (\lambda \epsilon \rightarrow (\rho l ==^L \text{unknown}) \longrightarrow \\ \text{alloc } \epsilon (\lambda \alpha \rightarrow \kappa (\rho [\alpha / l])), \\ \text{assign } (\rho l) \epsilon (\kappa \rho)) \end{aligned}$$


$$\mathcal{B}[\![\ (\text{begin } B^+) \ ]\!] \rho \kappa = \mathcal{B}^+[\![\ B^+ \ ]\!] \rho \kappa$$

-- Body sequence denotations  $\mathcal{B}^+[\![\ B^+ \ ]\!]$  :  $\mathbf{U} \rightarrow (\mathbf{U} \rightarrow \mathbf{C}) \rightarrow \mathbf{C}$ 

$$\mathcal{B}^+[\![\ \sqcup\ B \ ]\!] \rho \kappa = \mathcal{B}[\![\ B \ ]\!] \rho \kappa$$


$$\mathcal{B}^+[\![\ B \sqcup B^+ \ ]\!] \rho \kappa = \mathcal{B}[\![\ B \ ]\!] \rho (\lambda \rho' \rightarrow \mathcal{B}^+[\![\ B^+ \ ]\!] \rho' \kappa)$$

-- Program denotations  $\mathcal{P}[\![\ \Pi \ ]\!]$  :  $\mathbf{A}$ 

$$\mathcal{P}[\![\ \sqcup\sqcup\sqcup \ ]\!] = \text{finished initial-store}$$


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


```

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
```