

Assignment 6: Languages with contexts

Turn in your answer **through the submission page** (click on the link on the Assignments page) by 5:00pm on Friday December 5th. Send an email message as well to emsaad@cs.nmsu.edu after you have submitted your document.

1. Let the domain *Location* be *Nat*. (Thus, *first-locn* = *zero*, *next-locn* = ($\lambda l. l$ plus one), etc.) Simplify, as far as possible, the program

$\mathbf{P}[\mathbf{begin\ var\ } A; A := 2; \mathbf{begin\ var\ } B; B := A+1 \mathbf{end\ end.}]$, using language 1 as defined at the end of this assignment, with this modification.

2. Augment language 1 to include procedures:

$D ::= \dots \mid \mathbf{proc\ } I = C$

and procedure invocations:

$C ::= \dots \mid \mathbf{call\ } I$

Now augment the *Denotable-value* domain with the summand $Proc = Store \rightarrow Poststore_{\perp}$ to accommodate procedures. If the semantics of procedure definition is written as

$\mathbf{D}[\mathbf{proc\ } I = C] = \lambda e. (updateenv\ \llbracket I \rrbracket\ inProc(C\llbracket C \rrbracket e)\ e)$,

write the semantic equation for $\mathbf{C}[\mathbf{call\ } I]$. What kind of scoping is used? Explain your answer.

3. Using the semantics of the LETREC construct in language 2, determine the denotations of the following examples:

a. $\llbracket \mathbf{LET\ } G = \mathbf{LAMBDA\ } (X)\ X \mathbf{IN\ LET\ } G = \mathbf{LAMBDA\ } (Y)\ (G\ Y) \mathbf{IN\ } (G\ a_0) \rrbracket$

b. $\llbracket \mathbf{LETREC\ LISTIFY} = \mathbf{LAMBDA\ } (L)$
 $\mathbf{IFNULL\ } L \mathbf{THEN\ NIL}$
 $\mathbf{ELSE\ } (((\mathbf{HEAD\ } L) \mathbf{CONS\ NIL}) \mathbf{CONS\ } (\mathbf{LISTIFY\ } (\mathbf{TAIL\ } L)))$
 $\mathbf{IN\ LISTIFY\ } (a_0 \mathbf{CONS\ } (a_1 \mathbf{CONS\ NIL})) \rrbracket$

c. $\llbracket \mathbf{LETREC\ } L = \mathbf{HEAD\ } L \mathbf{IN\ } L \rrbracket$

Language 1

Abstract Syntax

$P \in \text{Program}$
 $K \in \text{Block}$
 $D \in \text{Declaration}$
 $C \in \text{Command}$
 $E \in \text{Expression}$
 $B \in \text{Boolean-expr}$
 $I \in \text{Identifier}$
 $N \in \text{Numeral}$
 $P ::= K.$
 $K ::= \text{begin } D; C \text{ end}$
 $D ::= D_1; D_2 \mid \text{var } I \mid \text{const } I = N$
 $C ::= C_1; C_2 \mid I := E \mid \text{while } B \text{ do } C \mid K$
 $E ::= E_1 + E_2 \mid I \mid N$
 $B ::= E_1 = E_2 \mid \sim B$

Semantic algebras

Domain Id (primitive, no operations)

Domain Nat (primitive)

Operations

$zero: Nat$

$one: Nat$

...

$plus: Nat \rightarrow Nat \rightarrow Nat$

$equals: Nat \rightarrow Nat \rightarrow Tr$

Domain Tr (primitive)

Operations

$true: Tr$

$false: Tr$

$(_ \rightarrow _ \square _): Tr \rightarrow A \rightarrow A \rightarrow A$

$not: Tr \rightarrow Tr$

$not(true) = false$

$not(false) = true$

Domain $Location$ (primitive)

Operations

$first-locn: Location$

$next-locn: Location \rightarrow Location$

$equal-locn : Location \rightarrow Location \rightarrow Tr$

$lessthan-locn : Location \rightarrow Location \rightarrow Tr$

Domain $Denotable-value = Location + Nat + Errvalue$
where $Errvalue = Unit$

Domain Environment = $(Id \rightarrow Denotable-value) \times Location$

Operations

$emptyenv : Location \rightarrow Environment$

$emptyenv = \lambda l. [(\lambda i. inErrorvalue()), l]$

$accessenv : Id \rightarrow Environment \rightarrow Storable-value$

$accessenv = \lambda i. \lambda [map, l]. map(i)$

$updateenv : Id \rightarrow Denotable-value \rightarrow Environment \rightarrow Environment$

$updateenv = \lambda i. \lambda d. \lambda [map, l]. [[i \mapsto d] map, l]$

$reserve-locn : Environment \rightarrow (Location \times Environment)$

$reserve-locn = \lambda [map, l]. [l, [map, next-locn(l)]]$

Domain $Expressible-value = Nat + Errvalue$
where $Errvalue = Unit$

Domain $Storable-value = Nat$

Domain $Store = Location \rightarrow Storable-value$

Operations

$access : Location \rightarrow Store \rightarrow Storable-value$

$access = \lambda i. \lambda s. s(i)$

$update : Location \rightarrow Storable-value \rightarrow Store \rightarrow Store$

$update = \lambda i. \lambda v. \lambda s [i \mapsto v] s$

Domain $Poststore = OK + Err$
where $OK = Err = Store$

Operations

$return : Store \rightarrow Poststore$
 $return = \lambda s.inOK(s)$
 $signalerr : Store \rightarrow Poststore$
 $signalerr = \lambda s.inErr(s)$
 $check : (Store \rightarrow Poststore_{\perp}) \rightarrow (Poststore_{\perp} \rightarrow Poststore_{\perp})$
 $check f = \lambda p.cases p$ of
 $isOK(s) \rightarrow (f s)$
 $\square isErr(s) \rightarrow p$
 end

Valuation Functions

P: $Location \rightarrow Store \rightarrow Poststore_{\perp}$
 $P[[K.]] = \lambda l.K[[K]](emptyenv l)$
K: $Block \rightarrow Environment \rightarrow Store \rightarrow Poststore_{\perp}$
 $K[[begin D; C end]] = \lambda e.C[[C]](D[[D]]e)$
D: $Declaration \rightarrow Environment \rightarrow Environment$
 $D[[D_1; D_2]] = D[[D_2]] \circ D[[D_1]]$
 $D[[const I = N]] = updateenv [[I]] inNat(N[[N]])$
 $D[[var I]] = \lambda e.let [l', e'] = (reserve - locn e) in updateenv [[I]] inLocation(l') e'$
C: $Command \rightarrow Environment \rightarrow Store \rightarrow Poststore_{\perp}$
 $C[[C_1; C_2]] = \lambda e.(check C[[C_2]]e) \circ (C[[C_1]]e)$
 $C[[I := E]] = \lambda e.\lambda s.cases accessenv [[I]]e$ of
 $isLocation(l) \rightarrow (cases E[[E]]e s$ of
 $isNat(n) \rightarrow (return(update l n s))$
 $\square isErrorvalue() \rightarrow (signalerr s)$
 end)
 $isNat(n) \rightarrow (signalerr s)$
 $\square isErrorvalue() \rightarrow (signalerr s)$
 end

$$\mathbf{C}[\mathbf{while\ B\ do\ C}] = \lambda e. fix(\lambda f. \lambda s. cases\ \mathbf{B}[\mathbf{B}]e\ s\ of$$

$$\quad isTr(t) \rightarrow (t \rightarrow (check\ f) \circ (\mathbf{C}[\mathbf{C}]e) \sqcap return)(s)$$

$$\quad isErrorvalue() \rightarrow (signalerr\ s)$$

$$\quad end)$$

$$\mathbf{C}[\mathbf{K}] = \mathbf{K}[\mathbf{K}]$$

E: Expression \rightarrow Environment \rightarrow Store \rightarrow Expressible-value

$$\mathbf{E}[\mathbf{E}_1 + \mathbf{E}_2] = \lambda e. \lambda s. cases\ \mathbf{E}[\mathbf{E}_1]e\ s\ of$$

$$\quad isNat(n_1) \rightarrow (cases\ \mathbf{E}[\mathbf{E}_2]e\ s\ of$$

$$\quad \quad isNat(n_2) \rightarrow inNat(n_1\ plus\ n_2)$$

$$\quad \quad \sqcap isErrorvalue() \rightarrow inErrorvalue()$$

$$\quad \quad end)$$

$$\quad \sqcap isErrorvalue() \rightarrow inErrorvalue()$$

$$\quad end)$$

$$\mathbf{E}[\mathbf{I}] = \lambda e. \lambda s. cases\ (accessenv[\mathbf{I}]e)\ of$$

$$\quad isNat(n) \rightarrow inNat(n)$$

$$\quad \sqcap isLocation(l) \rightarrow inNat(access\ l\ s)$$

$$\quad \sqcap isErrorvalue() \rightarrow inErrorvalue()$$

$$\quad end)$$

$$\mathbf{E}[\mathbf{N}] = \lambda e. \lambda s. inNat(\mathbf{N}[\mathbf{N}])$$

B: Boolean-expr \rightarrow Environment \rightarrow Store \rightarrow (Tr + Errvalue)

$$\mathbf{B}[\mathbf{E}_1 = \mathbf{E}_2] = \lambda e. \lambda s. cases\ (\mathbf{E}[\mathbf{E}_1]e\ s)\ of$$

$$\quad isNat(n_1) \rightarrow cases\ (\mathbf{E}[\mathbf{E}_2]e\ s)\ of$$

$$\quad \quad isNat(n_2) \rightarrow n_1\ equals\ n_2$$

$$\quad \quad \sqcap isErrvalue() \rightarrow inErrvalue()$$

$$\quad \quad end)$$

$$\quad \sqcap isErrvalue() \rightarrow inErrvalue()$$

$$\quad end)$$

$$\mathbf{B}[\sim\mathbf{B}] = \lambda e. \lambda s. not(\mathbf{B}[\mathbf{B}]e\ s)$$

N: Numeral \rightarrow Nat

$$\mathbf{N}[0] = zero$$

$$\mathbf{N}[1] = one$$

etc.

Language 2

Abstract Syntax

$E \in \text{Expression}$

$A \in \text{Atomic-symbol}$

$I \in \text{Identifier}$

$E ::= \text{LET } I = E_1 \text{ IN } E_2 \mid \text{LAMBDA } (I) E \mid E_1 E_2 \mid$
 $\text{IFNULL } E_1 \text{ THEN } E_2 \text{ ELSE } E_3 \mid \text{LETREC } I = E_1 \text{ IN } E_2 \mid$
 $I \mid A \mid (E) \mid$
 $E_1 \text{ CONS } E_2 \mid \text{HEAD } E \mid \text{TAIL } E \mid \text{NIL}$

Semantic algebras

Domain $Atom = (\text{infinite set of distinguishable objects})$

Domain $Denotable\text{-value} = (Function + List + Atom + Error)_{\perp}$

where $Function = Denotable\text{-value} \rightarrow Denotable\text{-value}$,

$List = Denotable\text{-value}^*$,

$Error = Unit$

Domain $Environment = Id \rightarrow Denotable\text{-value}$

$accessenv : Id \rightarrow Environment \rightarrow Denotable\text{-value}$

$accessenv = \lambda i. \lambda e. e(i)$

$updateenv : Id \rightarrow Denotable\text{-value} \rightarrow Environment \rightarrow Environment$

$updateenv = \lambda i. \lambda d. \lambda e. [i \mapsto d]e$

Domain $Denotable\text{-value}^*$

Operations

$nil : Denotable\text{-value}$

$nil = \text{inUnit}()$

$hd : Denotable\text{-value}^* \rightarrow Denotable\text{-value}$

$hd[x, [\dots]] = x$

$tl : Denotable\text{-value}^* \rightarrow Denotable\text{-value}^*$

$tl[x, y] = y$

$cons : Denotable\text{-value} \times Denotable\text{-value}^* \rightarrow Denotable\text{-value}^*$

$x \text{ cons } y = [x, y]$

Valuation functions

$E : \text{Expression} \rightarrow \text{Environment} \rightarrow \text{Expressible-value}$

where *Expressible-value* = *Denotable-value*

$E[\text{LET } I = E_1 \text{ IN } E_2] = \lambda e. E[E_2](\text{updateenv}[I](E[E_1]e)e)$

$E[\text{LAMBDA } (I) E] = \lambda e. \text{inFunction}(\lambda d. E[E](\text{updateenv}[I]d e))$

$E[E_1 E_2] = \lambda e. \text{cases } E[E_1]e \text{ of}$

$\text{isFunction}(f) \rightarrow f(E[E_2]e)$

$\square \text{isList}(t) \rightarrow \text{inError}()$

$\square \text{isAtom}(a) \rightarrow \text{inError}()$

$\square \text{isError}() \rightarrow \text{inError}()$

end

$E[\text{IFNULL } E_1 \text{ THEN } E_2 \text{ ELSE } E_3] =$

$\lambda e. \text{let } x = (E[E_1]e) \text{ in}$

cases x of

$\text{isFunction}(f) \rightarrow \text{inError}()$

$\square \text{isList}(t) \rightarrow ((\text{null } t)) \rightarrow (E[E_2]e \square E[E_3]e)$

$\square \text{isAtom}(a) \rightarrow \text{inError}()$

$\square \text{isError}() \rightarrow \text{inError}()$

end

$E[\text{LETREC } I = E_1 \text{ IN } E_2] = \lambda e. E[E_2](\text{fix } (\lambda e'. \text{updateenv}[I](E[E_1]e')e))$

$E[E_1 \text{ CONS } E_2] = \lambda e. \text{let } x = E[E_2]e \text{ in}$

cases x of

$\text{isFunction}(f) \rightarrow \text{inError}()$

$\square \text{isList}(t) \rightarrow \text{inList}((E[E_1]e) \text{ cons } t)$

$\square \text{isAtom}(a) \rightarrow \text{inError}()$

$\square \text{inError}() \rightarrow \text{inError}()$

end

$$\mathbf{E}[\text{HEAD } E] = \lambda e. \text{let } x = \mathbf{E}[E] e \text{ in}$$

cases x of

- $\text{isFunction}(f) \rightarrow \text{inError}()$
- $\square \text{isList}(t) \rightarrow (\text{null } t \rightarrow \text{nError}() \square (\text{hd } x))$
- $\square \text{isAtom}(a) \rightarrow \text{inError}()$
- $\square \text{inError}() \rightarrow \text{inError}()$

end

$$\mathbf{E}[\text{NIL}] = \lambda e. \text{inList}(\text{nil})$$

$$\mathbf{E}[\text{I}] = \text{accessenv}[\text{I}]$$

$$\mathbf{E}[\text{A}] = \lambda e. \text{inAtom}(\mathbf{A}[\text{A}])$$

$$\mathbf{E}[(E)] = \mathbf{E}[E]$$