

This section provides a formal denotational semantics for the primitive expressions of Scheme and selected built-in procedures.

$\langle \dots \rangle$ sequence formation
 $\#s$ length of sequence s
 $s \S t$ concatenation of sequences s and t
 $t \rightarrow a, b$ McCarthy conditional “if t then a else b ”
 $\rho[x/i]$ substitution “ ρ with x for i ”
 x in D injection of x into domain D
 $x | D$ projection of x to domain D

7.2.3 Semantic functions

$\mathcal{K} : \text{Con} \rightarrow \mathbf{E}$

$\mathcal{E} : \text{Exp} \rightarrow \mathbf{U} \rightarrow \mathbf{K} \rightarrow \mathbf{C}$

$\mathcal{E}^* : \text{Exp}^* \rightarrow \mathbf{U} \rightarrow \mathbf{K} \rightarrow \mathbf{C}$

$\mathcal{C} : \text{Com}^* \rightarrow \mathbf{U} \rightarrow \mathbf{C} \rightarrow \mathbf{C}$

Definition of \mathcal{K} deliberately omitted.

$\mathcal{E}[\mathbf{K}] =$
 $\lambda\rho\kappa.\text{send}(\mathcal{K}[\mathbf{K}], \kappa)$

$\mathcal{E}[\mathbf{I}] =$
 $\lambda\rho\kappa.\text{hold}(\text{lookup}(\rho, \mathbf{I}), \text{single}(\lambda\varepsilon. \text{if } \varepsilon = \text{undefined}$
then wrong(“undefined variable”)
else send(ε, κ)
endif))

$\mathcal{E}[(E_0 E^*)] =$
 $\lambda\rho\kappa.\mathcal{E}^*[\text{permute}(\langle E_0 \rangle \S E^*)](\rho, \lambda\varepsilon^*.\lambda\varepsilon^*_1.\text{apply}(\varepsilon^*_1 \downarrow 1, \varepsilon^*_1 \uparrow 1)(\kappa)(\text{unpermute}(\varepsilon^*)))$

$\mathcal{E}[(\text{lambda } (I^*) \Gamma^* E_0)] =$
 $\lambda\rho\kappa.\lambda\sigma. \text{if new } \sigma \in \mathbf{L}$
then send($\langle \langle \text{new } \sigma | \mathbf{L}, \lambda\varepsilon^* \kappa'.$ **if** $\# \varepsilon^* = \# I^*$
then tievals($\lambda\alpha^*.\lambda\rho'.\mathcal{C}[\Gamma^*](\rho', \mathcal{E}[E_0](\rho', \kappa'))(\text{extends}(\rho, I^*, \alpha^*)), \varepsilon^*$)
else wrong(“wrong number of arguments”)
endif)) in $\mathbf{E}, \kappa)(\text{update}(\text{new } \sigma | \mathbf{L}, \text{unspecified}, \sigma))$
else wrong(“out of memory”)(σ)
endif

$\mathcal{E}[(\text{lambda } (I^* . I) \Gamma^* E_0)] =$
 $\lambda\rho\kappa.\lambda\sigma. \text{if new } \sigma \in \mathbf{L}$
then send($\langle \langle \text{new } \sigma | \mathbf{L}, \lambda\varepsilon^* \kappa'.$ **if** $\# \varepsilon^* \geq \# I^*$
then tievalsrest($\lambda\alpha^*.\lambda\rho'.\mathcal{C}[\Gamma^*](\rho', \mathcal{E}[E_0](\rho', \kappa'))(\text{extends}(\rho, I^* \S \langle I \rangle, \alpha^*)), \varepsilon^*, \# I^*$)
else wrong(“too few arguments”)
endif)) in $\mathbf{E}, \kappa)(\text{update}(\text{new } \sigma | \mathbf{L}, \text{unspecified}, \sigma))$
else wrong(“out of memory”)(σ)
endif

$\mathcal{E}[(\text{lambda } I \Gamma^* E_0)] =$
 $\mathcal{E}[(\text{lambda } (\langle \rangle . I) \Gamma^* E_0)]$

$\mathcal{E}[(\text{if } E_0 E_1 E_2)] =$
 $\lambda\rho\kappa.\mathcal{E}[E_0](\rho, \text{single}(\lambda\varepsilon. \text{if } \text{truish}(\varepsilon)$
then $\mathcal{E}[E_1](\rho, \kappa)$
else $\mathcal{E}[E_2](\rho, \kappa)$
endif))

$$\begin{aligned} \mathcal{E}[\text{if } E_0 \ E_1] &= \\ \lambda\rho\kappa.\mathcal{E}[E_0](\rho, \text{single}(\lambda\varepsilon. &\text{if } \text{truish}(\varepsilon) \\ &\text{then } \mathcal{E}[E_1](\rho, \kappa) \\ &\text{else } \text{send}(\text{unspecified}, \kappa) \\ &\text{endif })) \end{aligned}$$

Here and elsewhere, any expressed value other than *undefined* may be used in place of *unspecified*.

$$\begin{aligned} \mathcal{E}[\text{set! } I \ E] &= \\ \lambda\rho\kappa.\mathcal{E}[E](\rho, \text{single}(\lambda\varepsilon. &\text{assign}(\text{lookup}(\rho, I), \varepsilon, \text{send}(\text{unspecified}, \kappa)))) \end{aligned}$$

$$\begin{aligned} \mathcal{E}^*[\] &= \\ \lambda\rho\kappa.\kappa(\) & \end{aligned}$$

$$\begin{aligned} \mathcal{E}^*[E_0 \ E^*] &= \\ \lambda\rho\kappa.\mathcal{E}[E_0](\rho, \text{single}(\lambda\varepsilon_0. &\mathcal{E}^*[E^*](\rho, \lambda\varepsilon^*.\kappa((\varepsilon_0)\ \&varepsilon^*)))) \end{aligned}$$

$$\begin{aligned} \mathcal{C}[\] &= \\ \lambda\rho\theta.\theta & \end{aligned}$$

$$\begin{aligned} \mathcal{C}[\Gamma_0 \ \Gamma^*] &= \\ \lambda\rho\theta.\mathcal{E}[\Gamma_0](\rho, \lambda\varepsilon^*. &\mathcal{C}[\Gamma^*](\rho, \theta)) \end{aligned}$$

7.2.4 Auxiliary functions

$$\begin{aligned} \text{lookup} &= \\ \lambda\rho I.\rho(I) & \end{aligned}$$

$$\begin{aligned} \text{extends} &= \\ \lambda\rho I^* \alpha^*. &\text{if } \#I^* = 0 \\ &\text{then } \rho \\ &\text{else } \text{extends}(\rho[\alpha^* \downarrow 1 / I^* \downarrow 1], I^* \uparrow 1, \alpha^* \uparrow 1) \\ &\text{endif} \end{aligned}$$

$$\text{wrong} : X \rightarrow C \quad [\text{implementation-dependent}]$$

$$\begin{aligned} \text{send} &= \\ \lambda\varepsilon\kappa.\kappa(\varepsilon) & \end{aligned}$$

$$\begin{aligned} \text{single} &= \\ \lambda\psi.\lambda\varepsilon^*. &\text{if } \#\varepsilon^* = 1 \\ &\text{then } \psi(\varepsilon^* \downarrow 1) \\ &\text{else } \text{wrong}(\text{"wrong number of return values"}) \\ &\text{endif} \end{aligned}$$

$$\text{new} : S \rightarrow (L + \{\text{error}\}) \quad [\text{implementation-dependent}]$$

$$\begin{aligned} \text{hold} &= \\ \lambda\alpha\kappa.\lambda\sigma. &\text{send}(\sigma(\alpha) \downarrow 1, \kappa)(\sigma) \end{aligned}$$

$$\begin{aligned} \text{assign} &= \\ \lambda\alpha\varepsilon\theta.\lambda\sigma. &\theta(\text{update}(\alpha, \varepsilon, \sigma)) \end{aligned}$$

$$\begin{aligned} \text{update} &= \\ \lambda\alpha\varepsilon\sigma.\sigma[&(\varepsilon, \text{true}) / \alpha] \end{aligned}$$

```

tievals =
λψε*.λσ. if #ε* = 0
  then ψ(⟨⟩)(σ)
  else if new σ ∈ L
    then tievals(λα*.ψ(⟨new σ | L⟩§α*), ε* † 1)(update(new σ | L, ε* ↓ 1, σ))
    else wrong("out of memory")(σ)
  endif
endif

```

```

tievalsrest =
λψε*ν.list(dropfirst(ε*, ν), single(λε.tievals(ψ, takefirst(ε*, ν)§⟨ε⟩)))

```

```

dropfirst =
λln. if n = 0
  then l
  else dropfirst(l † 1, n - 1)
endif

```

```

takefirst =
λln. if n = 0
  then ⟨⟩
  else ⟨l ↓ 1⟩§takefirst(l † 1, n - 1)
endif

```

```

truish =
λε. if ε = false
  then false
  else true
endif

```

permute : Exp* → Exp* [implementation-dependent]

unpermute : E* → E* [inverse of *permute*]

```

apply =
λεε*.λκ. if ε ∈ F
  then ε ↓ 2 | F(ε*, κ)
  else wrong("bad procedure")
endif

```

```

onearg =
λζ.λε*κ. if #ε* = 1
  then ζ(ε* ↓ 1, κ)
  else wrong("wrong number of arguments")
endif

```

```

twoarg =
λζ.λε*κ. if #ε* = 2
  then ζ(ε* ↓ 1, ε* ↓ 2, κ)
  else wrong("wrong number of arguments")
endif

```

```

list =
λε*κ. if #ε* = 0
  then send(null, κ)
  else list(ε* † 1, single(λε.cons(⟨ε* ↓ 1, ε⟩, κ)))
endif

```

```

cons =
twoarg( $\lambda\varepsilon_1\varepsilon_2\kappa.\lambda\sigma.$  if new  $\sigma \in L$ 
      then  $\lambda\sigma'.$  if new  $\sigma' \in L$ 
          then  $send(((new\ \sigma \mid L, new\ \sigma' \mid L, true)) \text{ in } E, \kappa)(update(new\ \sigma' \mid L, \varepsilon_2, \sigma'))$ 
          else  $wrong("out\ of\ memory", \sigma')$ 
          endif ( $update(new\ \sigma \mid L, \varepsilon_1, \sigma)$ )
      else  $wrong("out\ of\ memory")(\sigma)$ 
      endif )

```

```

less =
twoarg( $\lambda\varepsilon_1\varepsilon_2\kappa.$  if  $\varepsilon_1 \in R \wedge \varepsilon_2 \in R$ 
      then  $send(\text{if } \varepsilon_1 \mid R < \varepsilon_2 \mid R$ 
          then true
          else false
          endif ,  $\kappa)$ 
      else  $wrong("non-numeric\ argument\ to\ <")$ 
      endif )

```

```

add =
twoarg( $\lambda\varepsilon_1\varepsilon_2\kappa.$  if  $\varepsilon_1 \in R \wedge \varepsilon_2 \in R$ 
      then  $send((\varepsilon_1 \mid R + \varepsilon_2 \mid R) \text{ in } E, \kappa)$ 
      else  $wrong("non-numeric\ argument\ to\ +")$ 
      endif )

```

```

car =
onearg( $\lambda\varepsilon\kappa.$  if  $\varepsilon \in E_p$ 
      then  $hold(\varepsilon \mid E_p \downarrow 1, \kappa)$ 
      else  $wrong("non-pair\ argument\ to\ 'car'")$ 
      endif )

```

```

cdr =
onearg( $\lambda\varepsilon\kappa.$  if  $\varepsilon \in E_p$ 
      then  $hold(\varepsilon \mid E_p \downarrow 2, \kappa)$ 
      else  $wrong("non-pair\ argument\ to\ 'cdr'")$ 
      endif )

```

```

setcar =
twoarg( $\lambda\varepsilon_1\varepsilon_2\kappa.$  if  $\varepsilon_1 \in E_p$ 
      then if  $\varepsilon_1 \mid E_p \downarrow 3$ 
          then  $assign(\varepsilon_1 \mid E_p \downarrow 1, \varepsilon_2, send(unspecified, \kappa))$ 
          else  $wrong("immutable\ argument\ to\ 'set-car!')$ 
          endif
      else  $wrong("non-pair\ argument\ to\ 'set-car!')$ 
      endif )

```

```

ds:ev =
twoarg( $\lambda\varepsilon_1\varepsilon_2\kappa.$  if  $\varepsilon_1 \in M \wedge \varepsilon_2 \in M$ 
      then  $send(\text{if } \varepsilon_1 \mid M = \varepsilon_2 \mid M$ 
          then true
          else false
          endif ,  $\kappa)$ 
      else if  $\varepsilon_1 \in Q \wedge \varepsilon_2 \in Q$ 
          then  $send(\text{if } \varepsilon_1 \mid Q = \varepsilon_2 \mid Q$ 
              then true
              else false
          endif )
      endif )

```

```

        endif ,  $\kappa$ )
    else if  $\varepsilon_1 \in H \wedge \varepsilon_2 \in H$ 
        then send( if  $\varepsilon_1 \mid H = \varepsilon_2 \mid H$ 
            then true
            else false
            endif ,  $\kappa$ )
    else if  $\varepsilon_1 \in R \wedge \varepsilon_2 \in R$ 
        then send( if  $\varepsilon_1 \mid R = \varepsilon_2 \mid R$ 
            then true
            else false
            endif ,  $\kappa$ )
    else if  $\varepsilon_1 \in E_p \wedge \varepsilon_2 \in E_p$ 
        then send( $\lambda p_1 p_2.$  if  $ds:location-eq?(p_1 \downarrow 1, p_2 \downarrow 1) \wedge ds:location-eq?(p_1 \downarrow 2, p_2 \downarrow 2)$ 
            then true
            else false
            endif ( $\varepsilon_1 \mid E_p, \varepsilon_2 \mid E_p$ ),  $\kappa$ )
    else if  $\varepsilon_1 \in E_s \wedge \varepsilon_2 \in E_s$ 
        then send( if  $\varepsilon_1 \mid E_s = \varepsilon_2 \mid E_s$ 
            then true
            else false
            endif ,  $\kappa$ )
    else if  $\varepsilon_1 \in F \wedge \varepsilon_2 \in F$ 
        then send( if  $ds:location-eq?(\varepsilon_1 \mid F \downarrow 1, \varepsilon_2 \mid F \downarrow 1)$ 
            then true
            else false
            endif ,  $\kappa$ )
        else send(false,  $\kappa$ )
        endif
    endif
endif
endif
endif
endif )

```

```

apply =
twoarg( $\lambda \varepsilon_1 \varepsilon_2 \kappa.$  if  $\varepsilon_1 \in F$ 
    then valueslist( $\langle \varepsilon_2 \rangle, \lambda \varepsilon^*.apply(\varepsilon_1, \varepsilon^*)(\kappa)$ )
    else wrong("bad procedure argument to apply")
    endif )

```

```

valueslist =
onearg( $\lambda \varepsilon \kappa.$  if  $\varepsilon \in E_p$ 
    then cdr( $\langle \varepsilon \rangle, \lambda \varepsilon^*.valueslist(\varepsilon^*, \lambda \varepsilon^*_1.car(\langle \varepsilon \rangle, single(\lambda \varepsilon_1.\kappa(\langle \varepsilon_1 \rangle \S \varepsilon^*_1))))$ )
    else if  $\varepsilon = null$ 
        then  $\kappa(\langle \rangle)$ 
        else wrong("non-list argument to values-list")
        endif
    endif )

```

```

cwcs =
onearg( $\lambda \varepsilon \kappa.$  if  $\varepsilon \in F$ 
    then  $\lambda \sigma.$  if new  $\sigma \in L$ 
        then apply( $\varepsilon, \langle \langle new \sigma \mid L, \lambda \varepsilon^* \kappa'.\kappa(\varepsilon^*) \rangle \rangle$  in E)( $\kappa$ )(update(new  $\sigma \mid L, unspecified, \sigma$ ))
    endif
    endif )

```

```
        else wrong("out of memory", $\sigma$ )
        endif
else wrong("bad procedure argument")
endif )
```

```
values =
 $\lambda \varepsilon^* \kappa. \kappa(\varepsilon^*)$ 
```

```
cwv =
twoarg( $\lambda \varepsilon_1 \varepsilon_2 \kappa. \kappa.\text{apply}(\varepsilon_1, \langle \rangle)$ )( $\lambda \varepsilon^*. \kappa.\text{apply}(\varepsilon_2, \varepsilon^*)(\kappa)$ )
```