⟨quasiquotation D⟩ ⟶ `⟨qq template D⟩
  | (quasiquote ⟨qq template D⟩)
⟨qq template D⟩ ⟶ ⟨simple datum⟩
  | ⟨list qq template D⟩
  | ⟨vector qq template D⟩
  | ⟨unquotation D⟩
⟨list qq template D⟩ ⟶ (⟨qq template or splice D⟩*)
  | (⟨qq template or splice D⟩+ . ⟨qq template D⟩)
  | '⟨qq template D⟩
  | ⟨quasiquotation $D+1$⟩
⟨vector qq template D⟩ ⟶ #(⟨qq template or splice D⟩*)
⟨unquotation D⟩ ⟶ ,⟨qq template $D-1$⟩
  | (unquote ⟨qq template $D-1$⟩)
⟨qq template or splice D⟩ ⟶ ⟨qq template D⟩
  | ⟨splicing unquotation D⟩
⟨splicing unquotation D⟩ ⟶ ,@⟨qq template $D-1$⟩
  | (unquote-splicing ⟨qq template $D-1$⟩)

In ⟨quasiquotation⟩s, a ⟨list qq template D⟩ can sometimes be confused with either an ⟨unquotation D⟩ or a ⟨splicing unquotation D⟩. The interpretation as an ⟨unquotation⟩ or ⟨splicing unquotation D⟩ takes precedence.

7.1.5. Transformers

⟨transformer spec⟩ ⟶
  (syntax-rules (⟨identifier⟩*) ⟨syntax rule⟩*)
⟨syntax rule⟩ ⟶ (⟨pattern⟩ ⟨template⟩)
⟨pattern⟩ ⟶ ⟨pattern identifier⟩
  | (⟨pattern⟩*)
  | (⟨pattern⟩+ . ⟨pattern⟩)
  | (⟨pattern⟩* ⟨pattern⟩ ⟨ellipsis⟩)
  | #(⟨pattern⟩*)
  | #(⟨pattern⟩* ⟨pattern⟩ ⟨ellipsis⟩)
  | ⟨pattern datum⟩
⟨pattern datum⟩ ⟶ ⟨string⟩
  | ⟨character⟩
  | ⟨boolean⟩
  | ⟨number⟩
⟨template⟩ ⟶ ⟨pattern identifier⟩
  | (⟨template element⟩*)
  | (⟨template element⟩+ . ⟨template⟩)
  | #(⟨template element⟩*)
  | ⟨template datum⟩
⟨template element⟩ ⟶ ⟨template⟩
  | ⟨template⟩ ⟨ellipsis⟩
⟨template datum⟩ ⟶ ⟨pattern datum⟩
⟨pattern identifier⟩ ⟶ ⟨any identifier except ...⟩
⟨ellipsis⟩ ⟶ ⟨the identifier ...⟩

7.1.6. Programs and definitions

⟨program⟩ ⟶ ⟨command or definition⟩*
⟨command or definition⟩ ⟶ ⟨command⟩
  | ⟨definition⟩
  | ⟨syntax definition⟩
  | (begin ⟨command or definition⟩+)
⟨definition⟩ ⟶ (define ⟨variable⟩ ⟨expression⟩)
  | (define (⟨variable⟩ ⟨def formals⟩) ⟨body⟩)
  | (begin ⟨definition⟩*)
⟨def formals⟩ ⟶ ⟨variable⟩*
  | ⟨variable⟩* . ⟨variable⟩
⟨syntax definition⟩ ⟶
  (define-syntax ⟨keyword⟩ ⟨transformer spec⟩)

7.2. Formal semantics

This section provides a formal denotational semantics for the primitive expressions of Scheme and selected built-in procedures. The concepts and notation used here are described in [29]; the notation is summarized below:

$\langle \ldots \rangle$	sequence formation
$s \downarrow k$	kth member of the sequence s (1-based)
$\#s$	length of sequence s
$s \mathbin{\S} t$	concatenation of sequences s and t
$s \dagger k$	drop the first k members of sequence s
$t \rightarrow a, b$	McCarthy conditional "if t then a else b"
$\rho[x/i]$	substitution "ρ with x for i"
x in $\mathtt{D}$	injection of x into domain $\mathtt{D}$
$x \mid \mathtt{D}$	projection of x to domain $\mathtt{D}$

The reason that expression continuations take sequences of values instead of single values is to simplify the formal treatment of procedure calls and multiple return values.

The boolean flag associated with pairs, vectors, and strings will be true for mutable objects and false for immutable objects.

The order of evaluation within a call is unspecified. We mimic that here by applying arbitrary permutations *permute* and *unpermute*, which must be inverses, to the arguments in a call before and after they are evaluated. This is not quite right since it suggests, incorrectly, that the order of evaluation is constant throughout a program (for any given number of arguments), but it is a closer approximation to the intended semantics than a left-to-right evaluation would be.

The storage allocator *new* is implementation-dependent, but it must obey the following axiom: if $\mathit{new}\,\sigma \in \mathtt{L}$, then $\sigma\,(\mathit{new}\,\sigma \mid \mathtt{L}) \downarrow 2 = \mathit{false}$.

The definition of $\mathcal{K}$ is omitted because an accurate definition of $\mathcal{K}$ would complicate the semantics without being very interesting.

If P is a program in which all variables are defined before being referenced or assigned, then the meaning of P is

$$\mathcal{E}[\![\texttt{((lambda (I*) P')}\ \langle\text{undefined}\rangle\ \ldots\texttt{)}]\!]$$

where I^* is the sequence of variables defined in P, P′ is the sequence of expressions obtained by replacing every definition in P by an assignment, ⟨undefined⟩ is an expression that evaluates to *undefined*, and $\mathcal{E}$ is the semantic function that assigns meaning to expressions.

7.2.1. Abstract syntax

$\mathrm{K} \in \mathrm{Con}$	constants, including quotations
$\mathrm{I} \in \mathrm{Ide}$	identifiers (variables)
$\mathrm{E} \in \mathrm{Exp}$	expressions
$\Gamma \in \mathrm{Com} = \mathrm{Exp}$	commands

$$
\begin{array}{lll}
\mathrm{Exp} & \longrightarrow & \mathrm{K} \mid \mathrm{I} \mid \texttt{(}\mathrm{E}_0\ \mathrm{E}^*\texttt{)} \\
& \mid & \texttt{(lambda (}\mathrm{I}^*\texttt{)}\ \Gamma^*\ \mathrm{E}_0\texttt{)} \\
& \mid & \texttt{(lambda (}\mathrm{I}^*\ \texttt{.}\ \mathrm{I}\texttt{)}\ \Gamma^*\ \mathrm{E}_0\texttt{)} \\
& \mid & \texttt{(lambda}\ \mathrm{I}\ \Gamma^*\ \mathrm{E}_0\texttt{)} \\
& \mid & \texttt{(if}\ \mathrm{E}_0\ \mathrm{E}_1\ \mathrm{E}_2\texttt{)} \mid \texttt{(if}\ \mathrm{E}_0\ \mathrm{E}_1\texttt{)} \\
& \mid & \texttt{(set!}\ \mathrm{I}\ \mathrm{E}\texttt{)}
\end{array}
$$

7.2.2. Domain equations

$\alpha \in \mathtt{L}$		locations
$\nu \in \mathtt{N}$		natural numbers
$\mathtt{T}$	$= \{\mathit{false}, \mathit{true}\}$	booleans
$\mathtt{Q}$		symbols
$\mathtt{H}$		characters
$\mathtt{R}$		numbers
$\mathtt{E}_\mathrm{p}$	$= \mathtt{L} \times \mathtt{L} \times \mathtt{T}$	pairs
$\mathtt{E}_\mathrm{v}$	$= \mathtt{L}^* \times \mathtt{T}$	vectors
$\mathtt{E}_\mathrm{s}$	$= \mathtt{L}^* \times \mathtt{T}$	strings
$\mathtt{M}$	$= \{\mathit{false}, \mathit{true}, \mathit{null}, \mathit{undefined}, \mathit{unspecified}\}$	miscellaneous
$\phi \in \mathtt{F}$	$= \mathtt{L} \times (\mathtt{E}^* \to \mathtt{K} \to \mathtt{C})$	procedure values
$\epsilon \in \mathtt{E}$	$= \mathtt{Q} + \mathtt{H} + \mathtt{R} + \mathtt{E}_\mathrm{p} + \mathtt{E}_\mathrm{v} + \mathtt{E}_\mathrm{s} + \mathtt{M} + \mathtt{F}$	expressed values
$\sigma \in \mathtt{S}$	$= \mathtt{L} \to (\mathtt{E} \times \mathtt{T})$	stores
$\rho \in \mathtt{U}$	$= \mathrm{Ide} \to \mathtt{L}$	environments
$\theta \in \mathtt{C}$	$= \mathtt{S} \to \mathtt{A}$	command continuations
$\kappa \in \mathtt{K}$	$= \mathtt{E}^* \to \mathtt{C}$	expression continuations
$\mathtt{A}$		answers
$\mathtt{X}$		errors

7.2.3. Semantic functions

$$
\begin{aligned}
\mathcal{K} &: \mathrm{Con} \to \mathtt{E} \\
\mathcal{E} &: \mathrm{Exp} \to \mathtt{U} \to \mathtt{K} \to \mathtt{C} \\
\mathcal{E}^* &: \mathrm{Exp}^* \to \mathtt{U} \to \mathtt{K} \to \mathtt{C} \\
\mathcal{C} &: \mathrm{Com}^* \to \mathtt{U} \to \mathtt{C} \to \mathtt{C}
\end{aligned}
$$

Definition of $\mathcal{K}$ deliberately omitted.

$$\mathcal{E}[\![\mathrm{K}]\!] = \lambda\rho\kappa\,.\,\mathit{send}\,(\mathcal{K}[\![\mathrm{K}]\!])\,\kappa$$

$$
\begin{array}{l}
\mathcal{E}[\![\mathrm{I}]\!] = \lambda\rho\kappa\,.\,\mathit{hold}\,(\mathit{lookup}\,\rho\,\mathrm{I}) \\
\qquad\qquad (\mathit{single}(\lambda\epsilon\,.\,\epsilon = \mathit{undefined} \to \\
\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“undefined variable”}, \\
\qquad\qquad\qquad \mathit{send}\,\epsilon\,\kappa))
\end{array}
$$

$$
\begin{array}{l}
\mathcal{E}[\![\texttt{(}\mathrm{E}_0\ \mathrm{E}^*\texttt{)}]\!] = \\
\quad \lambda\rho\kappa\,.\,\mathcal{E}^*(\mathit{permute}(\langle\mathrm{E}_0\rangle \mathbin{\S} \mathrm{E}^*)) \\
\qquad\quad \rho \\
\qquad\quad (\lambda\epsilon^*\,.\,((\lambda\epsilon^*\,.\,\mathit{applicate}\,(\epsilon^* \downarrow 1)\,(\epsilon^* \dagger 1)\,\kappa) \\
\qquad\qquad\quad (\mathit{unpermute}\,\epsilon^*)))
\end{array}
$$

$$
\begin{array}{l}
\mathcal{E}[\![\texttt{(lambda (}\mathrm{I}^*\texttt{)}\ \Gamma^*\ \mathrm{E}_0\texttt{)}]\!] = \\
\quad \lambda\rho\kappa\,.\,\lambda\sigma\,. \\
\quad\quad \mathit{new}\,\sigma \in \mathtt{L} \to \\
\quad\quad\quad \mathit{send}\,(\langle \mathit{new}\,\sigma \mid \mathtt{L}, \\
\qquad\qquad\quad \lambda\epsilon^*\kappa'\,.\,\#\epsilon^* = \#\mathrm{I}^* \to \\
\qquad\qquad\qquad\quad \mathit{tievals}(\lambda\alpha^*\,.\,(\lambda\rho'\,.\,\mathcal{C}[\![\Gamma^*]\!]\rho'(\mathcal{E}[\![\mathrm{E}_0]\!]\rho'\kappa')) \\
\qquad\qquad\qquad\qquad\qquad (\mathit{extends}\,\rho\,\mathrm{I}^*\,\alpha^*)) \\
\qquad\qquad\qquad\qquad \epsilon^*, \\
\qquad\qquad\qquad\quad \mathit{wrong}\ \text{“wrong number of arguments”}\rangle \\
\qquad\qquad\quad \text{in}\ \mathtt{E}) \\
\qquad\qquad \kappa \\
\qquad\qquad (\mathit{update}\,(\mathit{new}\,\sigma \mid \mathtt{L})\,\mathit{unspecified}\,\sigma), \\
\quad\quad\quad \mathit{wrong}\ \text{“out of memory”}\ \sigma
\end{array}
$$

$$
\begin{array}{l}
\mathcal{E}[\![\texttt{(lambda (}\mathrm{I}^*\ \texttt{.}\ \mathrm{I}\texttt{)}\ \Gamma^*\ \mathrm{E}_0\texttt{)}]\!] = \\
\quad \lambda\rho\kappa\,.\,\lambda\sigma\,. \\
\quad\quad \mathit{new}\,\sigma \in \mathtt{L} \to \\
\quad\quad\quad \mathit{send}\,(\langle \mathit{new}\,\sigma \mid \mathtt{L}, \\
\qquad\qquad\quad \lambda\epsilon^*\kappa'\,.\,\#\epsilon^* \geq \#\mathrm{I}^* \to \\
\qquad\qquad\qquad\quad \mathit{tievalsrest} \\
\qquad\qquad\qquad\qquad (\lambda\alpha^*\,.\,(\lambda\rho'\,.\,\mathcal{C}[\![\Gamma^*]\!]\rho'(\mathcal{E}[\![\mathrm{E}_0]\!]\rho'\kappa')) \\
\qquad\qquad\qquad\qquad\qquad (\mathit{extends}\,\rho\,(\mathrm{I}^* \mathbin{\S} \langle\mathrm{I}\rangle)\,\alpha^*)) \\
\qquad\qquad\qquad\qquad \epsilon^* \\
\qquad\qquad\qquad\qquad (\#\mathrm{I}^*), \\
\qquad\qquad\qquad\quad \mathit{wrong}\ \text{“too few arguments”}\rangle\ \text{in}\ \mathtt{E}) \\
\qquad\qquad \kappa \\
\qquad\qquad (\mathit{update}\,(\mathit{new}\,\sigma \mid \mathtt{L})\,\mathit{unspecified}\,\sigma), \\
\quad\quad\quad \mathit{wrong}\ \text{“out of memory”}\ \sigma
\end{array}
$$

$$\mathcal{E}[\![\texttt{(lambda}\ \mathrm{I}\ \Gamma^*\ \mathrm{E}_0\texttt{)}]\!] = \mathcal{E}[\![\texttt{(lambda (.}\ \mathrm{I}\texttt{)}\ \Gamma^*\ \mathrm{E}_0\texttt{)}]\!]$$

$$
\begin{array}{l}
\mathcal{E}[\![\texttt{(if}\ \mathrm{E}_0\ \mathrm{E}_1\ \mathrm{E}_2\texttt{)}]\!] = \\
\quad \lambda\rho\kappa\,.\,\mathcal{E}[\![\mathrm{E}_0]\!]\,\rho\,(\mathit{single}\,(\lambda\epsilon\,.\,\mathit{truish}\,\epsilon \to \mathcal{E}[\![\mathrm{E}_1]\!]\rho\kappa, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{E}[\![\mathrm{E}_2]\!]\rho\kappa))
\end{array}
$$

$$
\begin{array}{l}
\mathcal{E}[\![\texttt{(if}\ \mathrm{E}_0\ \mathrm{E}_1\texttt{)}]\!] = \\
\quad \lambda\rho\kappa\,.\,\mathcal{E}[\![\mathrm{E}_0]\!]\,\rho\,(\mathit{single}\,(\lambda\epsilon\,.\,\mathit{truish}\,\epsilon \to \mathcal{E}[\![\mathrm{E}_1]\!]\rho\kappa, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{send}\,\mathit{unspecified}\,\kappa))
\end{array}
$$

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

$$
\begin{array}{l}
\mathcal{E}[\![\texttt{(set!}\ \mathrm{I}\ \mathrm{E}\texttt{)}]\!] = \\
\quad \lambda\rho\kappa\,.\,\mathcal{E}[\![\mathrm{E}]\!]\,\rho\,(\mathit{single}(\lambda\epsilon\,.\,\mathit{assign}\,(\mathit{lookup}\,\rho\,\mathrm{I}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \epsilon \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad (\mathit{send}\,\mathit{unspecified}\,\kappa)))
\end{array}
$$

$$\mathcal{E}^*[\![\,]\!] = \lambda\rho\kappa\,.\,\kappa\langle\,\rangle$$

$$
\begin{array}{l}
\mathcal{E}^*[\![\mathrm{E}_0\ \mathrm{E}^*]\!] = \\
\quad \lambda\rho\kappa\,.\,\mathcal{E}[\![\mathrm{E}_0]\!]\,\rho\,(\mathit{single}(\lambda\epsilon_0\,.\,\mathcal{E}^*[\![\mathrm{E}^*]\!]\,\rho\,(\lambda\epsilon^*\,.\,\kappa\,(\langle\epsilon_0\rangle \mathbin{\S} \epsilon^*))))
\end{array}
$$

$$\mathcal{C}[\![\,]\!] = \lambda\rho\theta\,.\,\theta$$

$$\mathcal{C}[\![\Gamma_0\ \Gamma^*]\!] = \lambda\rho\theta\,.\,\mathcal{E}[\![\Gamma_0]\!]\,\rho\,(\lambda\epsilon^*\,.\,\mathcal{C}[\![\Gamma^*]\!]\rho\theta)$$

7.2.4. Auxiliary functions

$$\begin{array}{l}
\mathit{lookup} : \mathtt{U} \to \mathrm{Ide} \to \mathtt{L} \\
\mathit{lookup} = \lambda\rho\mathrm{I}\,.\,\rho\mathrm{I}
\end{array}$$

$$\begin{array}{l}
\mathit{extends} : \mathtt{U} \to \mathrm{Ide}^* \to \mathtt{L}^* \to \mathtt{U} \\
\mathit{extends} = \\
\quad \lambda\rho\mathrm{I}^*\alpha^*\,.\,\#\mathrm{I}^* = 0 \to \rho, \\
\qquad\qquad\quad \mathit{extends}\,(\rho[(\alpha^* \downarrow 1)/(\mathrm{I}^* \downarrow 1)])\;(\mathrm{I}^* \dagger 1)\;(\alpha^* \dagger 1)
\end{array}$$

$\mathit{wrong} : \mathtt{X} \to \mathtt{C}$ [implementation-dependent]

$$\begin{array}{l}
\mathit{send} : \mathtt{E} \to \mathtt{K} \to \mathtt{C} \\
\mathit{send} = \lambda\epsilon\kappa\,.\,\kappa\langle\epsilon\rangle
\end{array}$$

$$\begin{array}{l}
\mathit{single} : (\mathtt{E} \to \mathtt{C}) \to \mathtt{K} \\
\mathit{single} = \\
\quad \lambda\psi\epsilon^*\,.\,\#\epsilon^* = 1 \to \psi(\epsilon^* \downarrow 1), \\
\qquad\qquad \mathit{wrong}\ \text{“wrong number of return values”}
\end{array}$$

$\mathit{new} : \mathtt{S} \to (\mathtt{L} + \{\mathit{error}\})$ [implementation-dependent]

$$\begin{array}{l}
\mathit{hold} : \mathtt{L} \to \mathtt{K} \to \mathtt{C} \\
\mathit{hold} = \lambda\alpha\kappa\sigma\,.\,\mathit{send}\,(\sigma\alpha \downarrow 1)\kappa\sigma
\end{array}$$

$$\begin{array}{l}
\mathit{assign} : \mathtt{L} \to \mathtt{E} \to \mathtt{C} \to \mathtt{C} \\
\mathit{assign} = \lambda\alpha\epsilon\theta\sigma\,.\,\theta(\mathit{update}\,\alpha\epsilon\sigma)
\end{array}$$

$$\begin{array}{l}
\mathit{update} : \mathtt{L} \to \mathtt{E} \to \mathtt{S} \to \mathtt{S} \\
\mathit{update} = \lambda\alpha\epsilon\sigma\,.\,\sigma[\langle\epsilon, \mathit{true}\rangle/\alpha]
\end{array}$$

$$\begin{array}{l}
\mathit{tievals} : (\mathtt{L}^* \to \mathtt{C}) \to \mathtt{E}^* \to \mathtt{C} \\
\mathit{tievals} = \\
\quad \lambda\psi\epsilon^*\sigma\,.\,\#\epsilon^* = 0 \to \psi\langle\,\rangle\sigma, \\
\qquad\qquad \mathit{new}\,\sigma \in \mathtt{L} \to \mathit{tievals}\,(\lambda\alpha^*\,.\,\psi(\langle \mathit{new}\,\sigma \mid \mathtt{L}\rangle \S \alpha^*)) \\
\qquad\qquad\qquad\qquad\qquad\qquad (\epsilon^* \dagger 1) \\
\qquad\qquad\qquad\qquad\qquad\qquad (\mathit{update}(\mathit{new}\,\sigma \mid \mathtt{L})(\epsilon^* \downarrow 1)\sigma), \\
\qquad\qquad\quad \mathit{wrong}\ \text{“out of memory”}\sigma
\end{array}$$

$$\begin{array}{l}
\mathit{tievalsrest} : (\mathtt{L}^* \to \mathtt{C}) \to \mathtt{E}^* \to \mathtt{N} \to \mathtt{C} \\
\mathit{tievalsrest} = \\
\quad \lambda\psi\epsilon^*\nu\,.\,\mathit{list}\,(\mathit{dropfirst}\,\epsilon^*\nu) \\
\qquad\qquad\qquad (\mathit{single}(\lambda\epsilon\,.\,\mathit{tievals}\,\psi\,((\mathit{takefirst}\,\epsilon^*\nu) \S \langle\epsilon\rangle)))
\end{array}$$

$\mathit{dropfirst} = \lambda l n\,.\,n = 0 \to l, \mathit{dropfirst}\,(l \dagger 1)(n-1)$

$\mathit{takefirst} = \lambda l n\,.\,n = 0 \to \langle\,\rangle, \langle l \downarrow 1\rangle \S (\mathit{takefirst}\,(l \dagger 1)(n-1))$

$$\begin{array}{l}
\mathit{truish} : \mathtt{E} \to \mathtt{T} \\
\mathit{truish} = \lambda\epsilon\,.\,\epsilon = \mathit{false} \to \mathit{false}, \mathit{true}
\end{array}$$

$\mathit{permute} : \mathrm{Exp}^* \to \mathrm{Exp}^*$ [implementation-dependent]

$\mathit{unpermute} : \mathtt{E}^* \to \mathtt{E}^*$ [inverse of *permute*]

$$\begin{array}{l}
\mathit{applicate} : \mathtt{E} \to \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{applicate} = \\
\quad \lambda\epsilon\epsilon^*\kappa\,.\,\epsilon \in \mathrm{F} \to (\epsilon \mid \mathrm{F} \downarrow 2)\epsilon^*\kappa, \mathit{wrong}\ \text{“bad procedure”}
\end{array}$$

$$\begin{array}{l}
\mathit{onearg} : (\mathtt{E} \to \mathtt{K} \to \mathtt{C}) \to (\mathtt{E}^* \to \mathtt{K} \to \mathtt{C}) \\
\mathit{onearg} = \\
\quad \lambda\zeta\epsilon^*\kappa\,.\,\#\epsilon^* = 1 \to \zeta(\epsilon^* \downarrow 1)\kappa, \\
\qquad\qquad \mathit{wrong}\ \text{“wrong number of arguments”}
\end{array}$$

$$\begin{array}{l}
\mathit{twoarg} : (\mathtt{E} \to \mathtt{E} \to \mathtt{K} \to \mathtt{C}) \to (\mathtt{E}^* \to \mathtt{K} \to \mathtt{C}) \\
\mathit{twoarg} = \\
\quad \lambda\zeta\epsilon^*\kappa\,.\,\#\epsilon^* = 2 \to \zeta(\epsilon^* \downarrow 1)(\epsilon^* \downarrow 2)\kappa, \\
\qquad\qquad \mathit{wrong}\ \text{“wrong number of arguments”}
\end{array}$$

$$\begin{array}{l}
\mathit{list} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{list} = \\
\quad \lambda\epsilon^*\kappa\,.\,\#\epsilon^* = 0 \to \mathit{send}\,\mathit{null}\,\kappa, \\
\qquad\qquad \mathit{list}\,(\epsilon^* \dagger 1)(\mathit{single}(\lambda\epsilon\,.\,\mathit{cons}\langle\epsilon^* \downarrow 1, \epsilon\rangle\kappa))
\end{array}$$

$$\begin{array}{l}
\mathit{cons} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{cons} = \\
\quad \mathit{twoarg}\,(\lambda\epsilon_1\epsilon_2\kappa\sigma\,.\,\mathit{new}\,\sigma \in \mathtt{L} \to \\
\qquad\qquad\qquad\qquad (\lambda\sigma'\,.\,\mathit{new}\,\sigma' \in \mathtt{L} \to \\
\qquad\qquad\qquad\qquad\qquad \mathit{send}\,(\langle \mathit{new}\,\sigma \mid \mathtt{L}, \mathit{new}\,\sigma' \mid \mathtt{L}, \mathit{true}\rangle \\
\qquad\qquad\qquad\qquad\qquad\quad \text{in } \mathtt{E}) \\
\qquad\qquad\qquad\qquad\qquad\quad \kappa \\
\qquad\qquad\qquad\qquad\qquad\quad (\mathit{update}(\mathit{new}\,\sigma' \mid \mathtt{L})\epsilon_2\sigma'), \\
\qquad\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“out of memory”}\sigma') \\
\qquad\qquad\qquad\qquad (\mathit{update}(\mathit{new}\,\sigma \mid \mathtt{L})\epsilon_1\sigma), \\
\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“out of memory”}\sigma)
\end{array}$$

$$\begin{array}{l}
\mathit{less} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{less} = \\
\quad \mathit{twoarg}\,(\lambda\epsilon_1\epsilon_2\kappa\,.\,(\epsilon_1 \in \mathtt{R} \wedge \epsilon_2 \in \mathtt{R}) \to \\
\qquad\qquad\qquad\qquad \mathit{send}\,(\epsilon_1 \mid \mathtt{R} < \epsilon_2 \mid \mathtt{R} \to \mathit{true}, \mathit{false})\kappa, \\
\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“non-numeric argument to <”})
\end{array}$$

$$\begin{array}{l}
\mathit{add} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{add} = \\
\quad \mathit{twoarg}\,(\lambda\epsilon_1\epsilon_2\kappa\,.\,(\epsilon_1 \in \mathtt{R} \wedge \epsilon_2 \in \mathtt{R}) \to \\
\qquad\qquad\qquad\qquad \mathit{send}\,((\epsilon_1 \mid \mathtt{R} + \epsilon_2 \mid \mathtt{R}) \text{ in } \mathtt{E})\kappa, \\
\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“non-numeric argument to +”})
\end{array}$$

$$\begin{array}{l}
\mathit{car} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{car} = \\
\quad \mathit{onearg}\,(\lambda\epsilon\kappa\,.\,\epsilon \in \mathrm{E_p} \to \mathit{hold}\,(\epsilon \mid \mathrm{E_p} \downarrow 1)\kappa, \\
\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“non-pair argument to }\mathtt{car}\text{”})
\end{array}$$

$\mathit{cdr} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C}$ [similar to *car*]

$$\begin{array}{l}
\mathit{setcar} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{setcar} = \\
\quad \mathit{twoarg}\,(\lambda\epsilon_1\epsilon_2\kappa\,.\,\epsilon_1 \in \mathrm{E_p} \to \\
\qquad\qquad\qquad\qquad (\epsilon_1 \mid \mathrm{E_p} \downarrow 3) \to \mathit{assign}\,(\epsilon_1 \mid \mathrm{E_p} \downarrow 1) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \epsilon_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\mathit{send}\,\mathit{unspecified}\,\kappa), \\
\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“immutable argument to }\mathtt{set\text{-}car!}\text{”}, \\
\qquad\qquad\qquad\qquad \mathit{wrong}\ \text{“non-pair argument to }\mathtt{set\text{-}car!}\text{”})
\end{array}$$

$$\begin{array}{l}
\mathit{eqv} : \mathtt{E}^* \to \mathtt{K} \to \mathtt{C} \\
\mathit{eqv} = \\
\quad \mathit{twoarg}\,(\lambda\epsilon_1\epsilon_2\kappa\,.\,(\epsilon_1 \in \mathtt{M} \wedge \epsilon_2 \in \mathtt{M}) \to \\
\qquad\qquad\qquad\qquad \mathit{send}\,(\epsilon_1 \mid \mathtt{M} = \epsilon_2 \mid \mathtt{M} \to \mathit{true}, \mathit{false})\kappa, \\
\qquad\qquad\qquad\quad (\epsilon_1 \in \mathtt{Q} \wedge \epsilon_2 \in \mathtt{Q}) \to \\
\qquad\qquad\qquad\qquad \mathit{send}\,(\epsilon_1 \mid \mathtt{Q} = \epsilon_2 \mid \mathtt{Q} \to \mathit{true}, \mathit{false})\kappa, \\
\qquad\qquad\qquad\quad (\epsilon_1 \in \mathtt{H} \wedge \epsilon_2 \in \mathtt{H}) \to \\
\qquad\qquad\qquad\qquad \mathit{send}\,(\epsilon_1 \mid \mathtt{H} = \epsilon_2 \mid \mathtt{H} \to \mathit{true}, \mathit{false})\kappa, \\
\qquad\qquad\qquad\quad (\epsilon_1 \in \mathtt{R} \wedge \epsilon_2 \in \mathtt{R}) \to \\
\qquad\qquad\qquad\qquad \mathit{send}\,(\epsilon_1 \mid \mathtt{R} = \epsilon_2 \mid \mathtt{R} \to \mathit{true}, \mathit{false})\kappa, \\
\qquad\qquad\qquad\quad (\epsilon_1 \in \mathrm{E_p} \wedge \epsilon_2 \in \mathrm{E_p}) \to \\
\qquad\qquad\qquad\qquad \mathit{send}\,((\lambda p_1 p_2\,.\,((p_1 \downarrow 1) = (p_2 \downarrow 1) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad (p_1 \downarrow 2) = (p_2 \downarrow 2)) \to \mathit{true}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{false}) \\
\qquad\qquad\qquad\qquad\qquad (\epsilon_1 \mid \mathrm{E_p}) \\
\qquad\qquad\qquad\qquad\qquad (\epsilon_2 \mid \mathrm{E_p})) \\
\qquad\qquad\qquad\qquad\quad \kappa,
\end{array}$$

$$
\begin{array}{l}
\qquad\qquad(\epsilon_1 \in \mathtt{E_v} \wedge \epsilon_2 \in \mathtt{E_v}) \rightarrow \ldots,\\
\qquad\qquad(\epsilon_1 \in \mathtt{E_s} \wedge \epsilon_2 \in \mathtt{E_s}) \rightarrow \ldots,\\
\qquad\qquad(\epsilon_1 \in \mathtt{F} \wedge \epsilon_2 \in \mathtt{F}) \rightarrow\\
\qquad\qquad\quad \mathit{send}\,((\epsilon_1 \mid \mathtt{F} \downarrow 1) = (\epsilon_2 \mid \mathtt{F} \downarrow 1) \rightarrow \mathit{true}, \mathit{false})\\
\qquad\qquad\qquad\quad \kappa,\\
\qquad\qquad\quad \mathit{send}\;\mathit{false}\,\kappa)
\end{array}
$$

$$
\begin{array}{l}
\mathit{apply} : \mathtt{E}^* \rightarrow \mathtt{K} \rightarrow \mathtt{C}\\
\mathit{apply} =\\
\quad \mathit{twoarg}\,(\lambda\epsilon_1\epsilon_2\kappa\,.\,\epsilon_1 \in \mathtt{F} \rightarrow \mathit{valueslist}\,\langle\epsilon_2\rangle(\lambda\epsilon^*\,.\,\mathit{applicate}\,\epsilon_1\epsilon^*\kappa),\\
\qquad\qquad\qquad\qquad \mathit{wrong}\text{ “bad procedure argument to }\mathtt{apply}\text{”})
\end{array}
$$

$$
\begin{array}{l}
\mathit{valueslist} : \mathtt{E}^* \rightarrow \mathtt{K} \rightarrow \mathtt{C}\\
\mathit{valueslist} =\\
\quad \mathit{onearg}\,(\lambda\epsilon\kappa\,.\,\epsilon \in \mathtt{E_p} \rightarrow\\
\qquad\qquad\qquad \mathit{cdr}\langle\epsilon\rangle\\
\qquad\qquad\qquad\quad (\lambda\epsilon^*\,.\,\mathit{valueslist}\\
\qquad\qquad\qquad\qquad\quad \epsilon^*\\
\qquad\qquad\qquad\qquad\quad (\lambda\epsilon^*\,.\,\mathit{car}\langle\epsilon\rangle(\mathit{single}(\lambda\epsilon\,.\,\kappa(\langle\epsilon\rangle \S \epsilon^*))))),\\
\qquad\qquad\quad \epsilon = \mathit{null} \rightarrow \kappa\langle\,\rangle,\\
\qquad\qquad\qquad \mathit{wrong}\text{ “non-list argument to }\mathtt{values\text{-}list}\text{”})
\end{array}
$$

$$
\begin{array}{l}
\mathit{cwcc} : \mathtt{E}^* \rightarrow \mathtt{K} \rightarrow \mathtt{C} \qquad [\mathtt{call\text{-}with\text{-}current\text{-}continuation}]\\
\mathit{cwcc} =\\
\quad \mathit{onearg}\,(\lambda\epsilon\kappa\,.\,\epsilon \in \mathtt{F} \rightarrow\\
\qquad\qquad\qquad (\lambda\sigma\,.\,\mathit{new}\,\sigma \in \mathtt{L} \rightarrow\\
\qquad\qquad\qquad\qquad\quad \mathit{applicate}\,\epsilon\\
\qquad\qquad\qquad\qquad\qquad\qquad \langle\langle\mathit{new}\,\sigma \mid \mathtt{L}, \lambda\epsilon^*\kappa'\,.\,\kappa\epsilon^*\rangle \text{ in } \mathtt{E}\rangle\\
\qquad\qquad\qquad\qquad\qquad\qquad \kappa\\
\qquad\qquad\qquad\qquad\qquad\qquad (\mathit{update}\,(\mathit{new}\,\sigma \mid \mathtt{L})\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{unspecified}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \sigma),\\
\qquad\qquad\qquad\qquad\quad \mathit{wrong}\text{ “out of memory” }\sigma),\\
\qquad\qquad\qquad \mathit{wrong}\text{ “bad procedure argument”})
\end{array}
$$

$$
\begin{array}{l}
\mathit{values} : \mathtt{E}^* \rightarrow \mathtt{K} \rightarrow \mathtt{C}\\
\mathit{values} = \lambda\epsilon^*\kappa\,.\,\kappa\epsilon^*
\end{array}
$$

$$
\begin{array}{l}
\mathit{cwv} : \mathtt{E}^* \rightarrow \mathtt{K} \rightarrow \mathtt{C} \qquad [\mathtt{call\text{-}with\text{-}values}]\\
\mathit{cwv} =\\
\quad \mathit{twoarg}\,(\lambda\epsilon_1\epsilon_2\kappa\,.\,\mathit{applicate}\,\epsilon_1\langle\,\rangle(\lambda\epsilon^*\,.\,\mathit{applicate}\,\epsilon_2\;\epsilon^*))
\end{array}
$$

7.3. Derived expression types

This section gives macro definitions for the derived expression types in terms of the primitive expression types (literal, variable, call, `lambda`, `if`, `set!`). See section 6.4 for a possible definition of `delay`.

```
(define-syntax cond
  (syntax-rules (else =>)
    ((cond (else result1 result2 ...))
     (begin result1 result2 ...))
    ((cond (test => result))
     (let ((temp test))
       (if temp (result temp))))
    ((cond (test => result) clause1 clause2 ...)
     (let ((temp test))
       (if temp
           (result temp)
           (cond clause1 clause2 ...))))
    ((cond (test)) test)
    ((cond (test) clause1 clause2 ...)
     (let ((temp test))
       (if temp
           temp
           (cond clause1 clause2 ...))))
    ((cond (test result1 result2 ...))
     (if test (begin result1 result2 ...)))
    ((cond (test result1 result2 ...)
           clause1 clause2 ...)
     (if test
         (begin result1 result2 ...)
         (cond clause1 clause2 ...)))))


(define-syntax case
  (syntax-rules (else)
    ((case (key ...)
       clauses ...)
     (let ((atom-key (key ...)))
       (case atom-key clauses ...)))
    ((case key
       (else result1 result2 ...))
     (begin result1 result2 ...))
    ((case key
       ((atoms ...) result1 result2 ...))
     (if (memv key '(atoms ...))
         (begin result1 result2 ...)))
    ((case key
       ((atoms ...) result1 result2 ...)
       clause clauses ...)
     (if (memv key '(atoms ...))
         (begin result1 result2 ...)
         (case key clause clauses ...)))))


(define-syntax and
  (syntax-rules ()
    ((and) #t)
    ((and test) test)
    ((and test1 test2 ...)
     (if test1 (and test2 ...) #f))))


(define-syntax or
  (syntax-rules ()
    ((or) #f)
    ((or test) test)
    ((or test1 test2 ...)
     (let ((x test1))
       (if x x (or test2 ...))))))


(define-syntax let
  (syntax-rules ()
    ((let ((name val) ...) body1 body2 ...)
     ((lambda (name ...) body1 body2 ...)
      val ...))
    ((let tag ((name val) ...) body1 body2 ...)
     ((letrec ((tag (lambda (name ...)
                      body1 body2 ...)))
        tag)
```