Daniel Marques

Alternative Denotational Semantics for F_0

$\mathcal{C}\,[[e]] : \text{Env} \rightarrow \text{Result}$
$\mathcal{C}_s\,[[e]] : \text{Env} \rightarrow \text{Value}_\perp$
$\mathcal{C}_l\,[[e]] : \text{Env} \rightarrow \text{Result}$
Result = Value + Thunk
Thunk = (Env → Result) ∗ Env
$\rho : \text{Env} \rightarrow \text{Result}$
Pair = Result ∗ Result
resolve : Result → Value $_\perp$
$\mathcal{C}_s\,[[e]] = \text{resolve}\,(\mathcal{C}_l\,[[e]]\,\rho)$
  resolve = fix $\lambda F \lambda r$.case $r$of
    Value $(v) \Longrightarrow v$
    Thunk $(t) \Longrightarrow$
      let $F = \pi_1 t,\ \rho = \pi_2 t$ in resolve $(f\,(\rho))$
$\mathcal{C}_l\,[[n]]\,\rho = n$
$\mathcal{C}_l\,[[x]]\,\rho = \rho(x)$
$\mathcal{C}_l\,[[e_0 \text{ binop } e_1]] = \text{case } \mathcal{C}_s\,[[e_0]] \text{ of Value}\,(v_1) \Longrightarrow \text{case } \mathcal{C}_s\,[[e_1]] \text{ of Value}\,(v_2) \Longrightarrow v_1 \text{ binop } v_2$
$\mathcal{C}_l\,[[\text{cons } e_0\ e1]]\,\rho \Longrightarrow \langle \mathcal{C}_l\,[[e_0]]\,\rho,\ \mathcal{C}_l\,[[e_1]]\,\rho \rangle$
$\mathcal{C}_l\,[[fn\ x\ e]]\,\rho = \lambda v.\, \mathcal{C}_l\,[[e]]\,\rho\,[x \mapsto v]$
$\mathcal{C}_l\,[[\text{call } e_0\ e_1]]\,\rho = \text{case } \mathcal{C}_s\,[[e_0]]\,\rho \text{ of Function } (f) \Longrightarrow f\,(\mathcal{C}_l\,[[e_1]]\,\rho)$
$\mathcal{C}_l\,[[\text{rec } x\ e]] = \text{fix } \lambda t.\ \langle \mathcal{C}_l\,[[e]]\,,\ \rho\,[x \mapsto t] \rangle$

Extending F_0 to Allow the Notion of State: the Language $F_!$

Idea: We have values that can change with time.
Identity is separate from its contents.
Functional Language → Imperative Language
Commands are allowed to mutate state.
Side effects must be specified.

Expressions in $F_!$

binop = +| − |∗
unop = zero? | unit? | hd | tl
expr = n | x | #t | #f | #u
  | (if e_0 e_1 e_2)
  | (unop e)
  | (e_0 binop e_1)
  | (cons e_0 e_1)
  | ($fn\ x\ e$)
  | (call e_0 e_1)
  | (seq $e_0 \cdots e_n$)
  | (ref e)   Creates a new location and initializes its value to e
  | (:= e_0 e_1)   Puts e_1 into location e_0
  | (ˆe)   Dereference. Gets the value in location e

The language F_0 supported both lazy and eager evaluation. The only difference was that eager evaluation might cause an infinite loop. With $F_!$ evaluation style (lazy or eager) effects the value of the computation, e.g.

(let (x (ref 0))
  (let (g (fn y (:= x 1)))
    (let (h ((fn z (^x)))
      (h (g 2)))))

with lazy evaluation result = 0
with eager evaluation result = 1

Operation Semantics for $F_!$ (Eager)

$\sigma \in$ Store
Store = Loc $\rightarrow$ Storable
Storable = Value
Value ::= n | #t | #f | #u | (cons v_1 v_2) | (fn x e) | (loc L)

$$\frac{\langle e_0,\, \sigma\rangle \rightarrow_1 \langle e_0',\, \sigma'\rangle}{\langle(\text{seq}\ e_0\ e_1),\, \sigma\rangle \rightarrow_1 \langle(\text{seq}\ e_0'\ e_1),\, \sigma'\rangle} \qquad \frac{}{\langle(\text{seq}\ v_0\ e_1),\, \sigma\rangle \rightarrow_1 \langle e_1,\, \sigma\rangle}$$

Note: Sequences of arbitrary length can be constructed as: (seq e_1(seq e_2($\cdots$(seq e_{n-1} e_n))))

$$\frac{\langle e_0,\, \sigma\rangle \rightarrow_1 \langle e_0',\, \sigma'\rangle}{\langle(\text{call}\ e_0\ e_1),\, \sigma\rangle \rightarrow_1 \langle(\text{call}\ e_0'\ e_1),\, \sigma'\rangle} \qquad \frac{\langle e_1,\, \sigma\rangle \rightarrow_1 \langle e_1',\, \sigma'\rangle}{\langle(\text{call}\ (fn\, x\, e_0)\ e_1),\, \sigma\rangle \rightarrow_1 \langle(\text{call}\ (fn\, x\, e_0)\ e_1'),\, \sigma'\rangle}$$

$$\frac{}{\langle(\text{call}\ (fn\, x\, e_0)\ v_1),\, \sigma\rangle \rightarrow_1 \langle e_0[x \mapsto v_1],\, \sigma\rangle}$$

$$\frac{\langle e,\, \sigma\rangle \rightarrow_1 \langle e',\, \sigma'\rangle}{\langle(\text{ref}\ e),\, \sigma\rangle \rightarrow_1 \langle(\text{ref}\ e'),\, \sigma'\rangle} \qquad \frac{}{\langle(\text{ref}\ v),\, \sigma\rangle \rightarrow_1 \langle(\text{loc L}),\, \sigma[L \mapsto v]\rangle}, \text{ where (L, v}') \notin \sigma$$

$$\frac{\langle e,\, \sigma\rangle \rightarrow_1 \langle e',\, \sigma'\rangle}{\langle(\hat{}\ e),\, \sigma\rangle \rightarrow_1 \langle(\hat{}\ e'),\, \sigma'\rangle} \qquad \frac{}{\langle(\hat{}(\text{loc L})),\, \sigma\rangle \rightarrow_1 \langle\sigma(\text{L}),\, \sigma\rangle}$$

$$\frac{\langle e_0,\, \sigma\rangle \rightarrow_1 \langle e_0',\, \sigma'\rangle}{\langle(:=\ e_0\ e_1),\, \sigma\rangle \rightarrow_1 \langle(:=\ e_0'\ e_1),\, \sigma'\rangle} \qquad \frac{\langle e_1,\, \sigma\rangle \rightarrow_1 \langle e_1',\, \sigma'\rangle}{\langle(:=\ (\text{loc L})\ e_1),\, \sigma\rangle \rightarrow_1 \langle(:=\ (\text{loc L})\ e_1'),\, \sigma'\rangle}$$

$$\frac{}{\langle(:=\ (\text{loc L})\ v),\, \sigma\rangle \rightarrow_1 \langle v,\, \sigma[L \mapsto v]\rangle}$$

$$\frac{}{\langle(\text{free}\ (\text{loc L})),\, \sigma\rangle \rightarrow_1 \langle\sigma(L),\, \sigma - (L, \sigma(L))\rangle}$$

$$\frac{}{\langle(\text{rec}\ x\ e),\, \sigma\rangle \rightarrow_1 \langle e[(\text{rec}\ x\ x)/x],\, \sigma\rangle}$$

$$\frac{\langle e_0,\, \sigma\rangle \rightarrow_1 \langle e_0',\, \sigma'\rangle}{\langle(\text{if}\ e_0\ e_1\ e_2),\, \sigma\rangle \rightarrow_1 \langle(\text{if}\ e_0'\ e_1\ e_2),\, \sigma'\rangle}$$

$$\frac{}{\langle(\text{if}\ \#\text{t}\ e_1\ e_2),\, \sigma\rangle \rightarrow_1 \langle e_1,\, \sigma\rangle} \qquad \frac{}{\langle(\text{if}\ \#\text{f}\ e_1\ e_2),\, \sigma\rangle \rightarrow_1 \langle e_2,\, \sigma\rangle}$$