1 source language $\mathcal{T}$

$$
\begin{aligned}
\texttt{n} ::= \quad & n \in \mathcal{N} \\
\texttt{e} ::= \quad & \texttt{n} \mid \texttt{true} \mid \texttt{false} \mid () \mid \texttt{x} \\
& \mid \texttt{e} + \texttt{e} \mid \texttt{e} - \texttt{e} \mid \texttt{e} * \texttt{e} \mid \texttt{e}/\texttt{e} \\
& \mid \texttt{e} = \texttt{e} \mid \texttt{e} < \texttt{e} \mid \texttt{e} >= \texttt{e} \mid \texttt{not e} \\
& \mid \texttt{if e then e else e} \\
& \mid \texttt{let x} = \texttt{e in e} \mid \texttt{let rec x x} = \texttt{e in e} \mid \texttt{fun x} \rightarrow \texttt{e} \mid \texttt{e e}
\end{aligned}
$$

2 Virtual Machine specificatoin

2.1 closure representation

$$Q = \langle I, C, F, Um \rangle$$

I is instruction sequense, C is constant table, F is closure table and Um is just an information, to create upvalues, composed of index and *meta* index, which of the former represents current register value and the latter does the upvalue index.

2.2 machine state

$$S = \langle Q, \texttt{pc}, U, R \rangle$$

`pc` is program counter which points nth instruction of I, U is upvalues table and R is register list.

2.3 values

$$\texttt{v} ::= \quad \texttt{null} \mid () \mid \texttt{i} \in \mathcal{N} \mid \texttt{true} \mid \texttt{false} \mid \texttt{clos}\,(Q, U)$$

2.4 Instructions

- `Load(a, kx)`

  set `C[kx]` to `R[a]`
- `SetBool(a, x, p)`

  set boolean `x > 0` to `R[a]`; if `p = 1` then `pc++`
- `Unit(a)`

  set `()` to `R[a]`
- `Clos(a, cx, p)`

  create $U' = U \ @ \ \{u \mid i_u \in Um \wedge u = R[i_u]\}$ and set $\langle \texttt{F[cx]}, 0, U', R' \rangle$ with environment to `R[a]` if `p=1` then the closure is recursive
- `Upval(a, ux)`

  set `U[rx]` to `R[a]`
- `Add(a, b, c)`

  set `R[b] + R[c]` to `R[a]`
- `Sub(a, b, c)`

  set `R[b] - R[c]` to `R[a]`
- `Mul(a, b, c)`

  set `R[b] * R[c]` to `R[a]`
- `Div(a, b, c)`

  set `R[b] / R[c]` to `R[a]`
- `Eq(a, b)`

  if `R[a] == R[b]` then `pc++`
- `Lt(a, b)`

  if `R[a] < R[b]` then `pc++`
- `Ge(a, b)`

  if `R[a] >= R[b]` then `pc++`

- `Test(a, p)`

  if `p > 0 && R[a] || p <= 0 && !R[a]` then `pc++`

- `Jump(x)`

  `pc += x`

- `Move(a, b)`

  set `R[b]` to `R[a]`

- `Call(a, b)`

  call closure `R[a]` with argument `R[b]` and set return value to `R[a]`

- `Return(a)`

  exit closure evaluation and return `R[a]`

- `TailCall(a, b)`

  call closure `R[a]` with argument `R[b]` and exit closure evaluation and return `R[a]`

3 compilation: translate source language program to VM initial state (input bytecode)

Compilation is represented as equation:

$$[\![\mathcal{T}, \Sigma, Q]\!] = Q', \Sigma'.$$

Σ is an environment from source language variable to register index. Q_{init} generates initial compilation state:

$$Q_{init}(e) = [\![e, [], \langle [], [], [], [] \rangle]\!].$$

3.1 auxiliary funcion

We define two auxiliary functions; *mtch* is to get last used register and rest instructions by pattern match,

$$\begin{array}{lcr} mtch\,(I; \texttt{Return}\,(\texttt{a})) & = & (\texttt{a}, I) \\ mtch\,(I; \texttt{TailCall}\,(\texttt{a}, \texttt{b})) & = & (\texttt{a}, I; \texttt{Call}\,(\texttt{a}, \texttt{b})) \\ mtch\,(\_) & = & \text{undefined} \end{array}$$

FV is to get free variables.

$$\begin{array}{lcr} FV\,(\texttt{n}) \mid FV\,(\texttt{true}) \mid FV\,(\texttt{false}) \mid FV\,(()) & = & \{\} \\ FV(\texttt{x}) & = & \{\texttt{x}\} \\ FV(\texttt{not e}) & = & FV(\texttt{e}) \\ FV(\texttt{e1?e2}) \mid FV\,(\texttt{e1 e2}) & = & FV(\texttt{e1}) \cup FV(\texttt{e2}) \\ FV(\texttt{if e1 then e2 else e3}) & = & FV(\texttt{e1}) \cup FV(\texttt{e2}) \cup FV(\texttt{e3}) \\ FV\,(\texttt{fun x} \rightarrow \texttt{e}) & = & FV\,(\texttt{e}) \setminus \{\texttt{x}\} \\ FV\,(\texttt{let x} = \texttt{e1 in e2}) & = & FV(\texttt{e1}) \cup (FV\,(\texttt{e2}) \setminus \{\texttt{x}\}) \\ FV\,(\texttt{let rec f x} = \texttt{e1 in e2}) & = & (FV\,(\texttt{e1}) \setminus \{\texttt{x}\} \cup FV\,(\texttt{e2})) \setminus \{\texttt{f}\} \end{array}$$

3.2 compilation rules

$$\frac{\texttt{a} \text{ and } \texttt{kx} \text{ are fresh}}{[\![\texttt{n}, \Sigma, \langle I, C, F, Um \rangle]\!] = \langle I; \texttt{Load}(\texttt{a}, \texttt{kx}); \texttt{Return}(\texttt{a}), C; \texttt{kx} = \texttt{n}, F, Um \rangle, \Sigma'} \quad (\textsc{Int})$$

$$[\![(), \Sigma, \langle I, C, F, Um \rangle]\!] = \langle I; \texttt{Unit}(\texttt{a}); \texttt{Return}(\texttt{a}), C, F, Um \rangle, \Sigma' \quad (\textsc{Unit})$$

$$[\![\texttt{true}, \Sigma, \langle I, C, F, Um \rangle]\!] = \langle I; \texttt{SetBool}(\texttt{a}, 1); \texttt{Return}(\texttt{a}), C, F, Um \rangle, \Sigma' \quad (\textsc{True})$$

$$[\![\texttt{false}, \Sigma, \langle I, C, F, Um\rangle]\!] = \langle I; \texttt{SetBool(a}, 0); \texttt{Return(a)}, C, F, Um\rangle, \Sigma' \tag{False}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1} + \texttt{e2}, \Sigma, Q]\!] = \langle I''; \texttt{Add(a}, \texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}); \texttt{Return(a)}, C'', F'', Um''\rangle, \Sigma''} \tag{Add}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1} - \texttt{e2}, \Sigma, Q]\!] = \langle I''; \texttt{Sub(a}, \texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}); \texttt{Return(a)}, C'', F'', Um''\rangle, \Sigma''} \tag{Sub}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1} * \texttt{e2}, \Sigma, Q]\!] = \langle I''; \texttt{Mul(a}, \texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}); \texttt{Return(a)}, C'', F'', Um''\rangle, \Sigma''} \tag{Mul}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1} / \texttt{e2}, \Sigma, Q]\!] = \langle I''; \texttt{Div(a}, \texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}); \texttt{Return(a)}, C'', F'', Um''\rangle, \Sigma''} \tag{Div}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1} = \texttt{e2}, \Sigma, Q]\!] = \langle I''; \texttt{Eq}(\texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}); \texttt{SetBool(a}, 1); \texttt{SetBool(a}, 0); \texttt{Return(a)}, C'', F'', Um''\rangle, \Sigma''} \tag{Eq}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1} < \texttt{e2}, \Sigma, Q]\!] = \langle I''; \texttt{Lt}(\texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}); \texttt{SetBool(a}, 1); \texttt{SetBool(a}, 0); \texttt{Return(a)}, C'', F'', Um''\rangle, \Sigma''} \tag{Lt}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1} >= \texttt{e2}, \Sigma, Q]\!] = \langle I''; \texttt{Ge}(\texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}); \texttt{SetBool(a}, 1); \texttt{SetBool(a}, 0); \texttt{Return(a)}, C'', F'', Um''\rangle, \Sigma''} \tag{Ge}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ is fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\texttt{e}, \Sigma, Q]\!] = \langle I_{\texttt{e}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e}}) = (\texttt{r}_{\texttt{e}}, I')\end{array}}{[\![\texttt{not e}, \Sigma, Q]\!] = \langle I'; \texttt{Test}(\texttt{r}_{\texttt{e}}, 0); \texttt{SetBool(a}, 1); \texttt{SetBool(a}, 0); \texttt{Return(a)}, C', F', Um'\rangle, \Sigma'} \tag{Not}$$

$$\frac{\begin{array}{c}\texttt{a} \text{ and } \texttt{cx} \text{ are fresh} \qquad Q = \langle I, C, F, Um\rangle \qquad fv = FV\,(\texttt{fun x} \to \texttt{e}) \qquad \Sigma_0 = \{(\texttt{x} = \texttt{i}_{\texttt{r}}) \mid (\texttt{x} = \texttt{i}_{\texttt{r}}) \in \Sigma \wedge \texttt{x} \in fv\} \\ \Sigma' = \{(\texttt{x} = -(\texttt{i} + 1)) \mid (\texttt{x} = \texttt{i}_{\texttt{r}}) \text{ as } xi \in \Sigma_0 \wedge xi \text{ is } i\text{th element of } \Sigma_0\} \qquad Um' = \{\texttt{i} \mid (\_ = \texttt{i}) \in \Sigma'\} \\ Q_{init}\,(\texttt{e}) = [\![\texttt{e}, \Sigma_{\texttt{e}}, \langle I_{\texttt{e}}, C_{\texttt{e}}, F_{\texttt{e}}, \_\rangle]\!] \qquad \Sigma'_{\texttt{e}} = \Sigma_{\texttt{e}}; \texttt{x} = 0 \qquad [\![\texttt{e}, \Sigma'_{\texttt{e}}, \langle I_{\texttt{e}}, C_{\texttt{e}}, F_{\texttt{e}}, Um'\rangle]\!] = \langle I'_{\texttt{e}}, C'_{\texttt{e}}, F'_{\texttt{e}}, \_\rangle, \Sigma''_{\texttt{e}} \\ Clos = \left\langle I'_{\texttt{e}}, C'_{\texttt{e}}, F'_{\texttt{e}}, Um @ Um'\right\rangle\end{array}}{[\![\texttt{fun x} \to \texttt{e}, \Sigma, Q]\!] = \langle I; \texttt{Clos(a}, \texttt{cx}, 0); \texttt{Return(a)}, C, F; \texttt{cx} = Clos, Um\rangle, \Sigma} \tag{Fun}$$

$$\frac{\begin{array}{c}Q = \langle I, C, F, Um\rangle \qquad [\![\texttt{e1}, \Sigma, Q]\!] = \langle I_{\texttt{e1}}, C', F', Um'\rangle, \Sigma' \qquad mtch\,(I_{\texttt{e1}}) = (\texttt{r}_{\texttt{e1}}, I') \\ [\![\texttt{e2}, \Sigma', \langle I', C', F', Um'\rangle]\!] = \langle I_{\texttt{e2}}, C'', F'', Um''\rangle, \Sigma'' \qquad mtch\,(I_{\texttt{e2}}) = (\texttt{r}_{\texttt{e2}}, I'')\end{array}}{[\![\texttt{e1 e2}, \Sigma, Q]\!] = \langle I''; \texttt{TailCall}(\texttt{r}_{\texttt{e1}}, \texttt{r}_{\texttt{e2}}), C'', F'', Um''\rangle, \Sigma''} \tag{Call}$$

$$\frac{\Sigma[\mathtt{x}] = \mathtt{i} \wedge \mathtt{i} \geq 0}{[\![\mathtt{x}, \Sigma, \langle I, C, F, Um\rangle]\!] = \langle I; \mathtt{Move}(\mathtt{a}, \mathtt{i}); \mathtt{Return}(\mathtt{a}), C, F, Um\rangle, \Sigma'} \quad (\textsc{VarLocal})$$

$$\frac{\Sigma[\mathtt{x}] = \mathtt{i} \wedge \mathtt{i} < 0}{[\![\mathtt{x}, \Sigma, \langle I, C, F, Um\rangle]\!] = \langle I; \mathtt{Upval}(\mathtt{a}, -\mathtt{i} - 1); \mathtt{Return}(\mathtt{a}), C, F, Um\rangle, \Sigma'} \quad (\textsc{VarUpval})$$

$$\frac{\begin{array}{c} \mathtt{a} \text{ and } \mathtt{b} \text{ are fresh} \qquad Q = \langle I, C, F, Um\rangle \\ [\![\mathtt{e}_1, \Sigma, Q]\!] = \left\langle I_{\mathtt{e1}}, C', F', Um'\right\rangle, \Sigma' \qquad mtch\,(I_{\mathtt{e1}}) = (\mathtt{r}_{\mathtt{e1}}, I') \\ [\![\mathtt{e}_2, \Sigma'; \mathtt{x} = \mathtt{a}, \left\langle I'; \mathtt{Move}(\mathtt{b}, \mathtt{a}), C', F', Um'\right\rangle]\!] = Q', \Sigma'' \end{array}}{[\![\mathtt{let\ x} = \mathtt{e}_1 \mathtt{\ in\ } \mathtt{e}_2, \Sigma, Q]\!] = Q', \Sigma''} \quad (\textsc{Let})$$

$$\frac{\begin{array}{c} \mathtt{a} \text{ and } \mathtt{cx} \text{ are fresh} \qquad Q = \langle I, C, F, Um\rangle \\ fv = (FV\,(\mathtt{e}_1) \setminus \{\mathtt{x}\} \cap FV\,(\mathtt{e}_2)) \setminus \{f\} \qquad \Sigma_0 = \{(\mathtt{x} = \mathtt{i}_\mathtt{r}) \mid (\mathtt{x} = \mathtt{i}_\mathtt{r}) \in \Sigma \wedge \mathtt{x} \in fv\} \cap \{(\mathtt{f} = \mathtt{a})\} \\ \Sigma' = \{(\mathtt{x} = -(\mathtt{i} + 1)) \mid (\mathtt{x} = \mathtt{i}_\mathtt{r}) \text{ as } xi \in \Sigma_0 \wedge xi \text{ is } i\text{th element of } \Sigma_0\} \\ Um' = \{\mathtt{i} \mid (\_ = \mathtt{i}) \in \Sigma'\} \qquad Q_{init}(\mathtt{e}_1) = [\![\mathtt{e1}, \Sigma_{\mathtt{e1}}, \langle I_{\mathtt{e1}}, C_{\mathtt{e1}}, F_{\mathtt{e1}}, \_\rangle]\!] \qquad \Sigma'_{\mathtt{e1}} = \Sigma_{\mathtt{e1}}; \mathtt{x} = 0 \\ [\![\mathtt{e1}, \Sigma'_{\mathtt{e1}}, \left\langle I_{\mathtt{e1}}, C_{\mathtt{e1}}, F_{\mathtt{e1}}, Um'\right\rangle]\!] = \langle I'_{\mathtt{e1}}, C'_{\mathtt{e1}}, F'_{\mathtt{e1}}, \_\rangle, \Sigma''_{\mathtt{e1}} \qquad Clos = \langle I_{\mathtt{e1}},', C'_{\mathtt{e1}}, F'_{\mathtt{e1}}, Um@Um'\rangle \\ I' = I; \mathtt{Clos}(\mathtt{a}, \mathtt{cx}, 1) \qquad F' = F; \mathtt{cx} = Clos \qquad \Sigma' = \Sigma; \mathtt{f} = \mathtt{a} \qquad [\![\mathtt{e2}, \Sigma', \langle I', C, F', Um\rangle]\!] = Q', \Sigma'' \end{array}}{[\![\mathtt{let\ rec\ f\ x} = \mathtt{e1\ in\ e2}, \Sigma, Q]\!] = Q', \Sigma''}$$

$(\textsc{LetRecFun})$