$$v, \Sigma \to v, \Sigma \ (\textsc{LVal})$$

$$\frac{\Sigma = (\Gamma, S)}{x, \Sigma \to S(\Gamma(x)), \Sigma} \ (\textsc{LVar})$$

$$\frac{\begin{array}{c} e_1, \Sigma_1 \to v_1, (\Gamma_2, S_2) \\ p \text{ fresh} \\ e_2, (\Gamma_2[x \leftarrow p], S_2[p \leftarrow v_1]) \to v_2, (\Gamma_3, S_3) \end{array}}{\text{let } x = e_1 \ \text{ in } e_2, \Sigma_1 \to v_2, (\Gamma_2, S_3)} \ (\textsc{LLet})$$

$$\frac{\begin{array}{c} e_1, \Sigma_1 \to \langle \Gamma_1, x, e_3 \rangle, \Sigma_2 \\ e_2, \Sigma_2 \to v_1, (\Gamma_3, S_3) \\ p \ \text{ fresh} \\ e_3, (\Gamma_1[x \leftarrow p], S_3[p \leftarrow v_1]) \to v_2, (\Gamma_4, S_4) \end{array}}{e_1(e_2), \Sigma_1 \to v_2, (\Gamma_3, S_4)} \ (\textsc{LApp})$$

$$\frac{e_1, \Sigma_1 \to v_1, (\Gamma, S) \qquad \Gamma(x) = p}{x := e, \Sigma \to v, (\Gamma, S[p \leftarrow v])} \ (\textsc{LAsn})$$

$$\frac{\Sigma = (\Gamma, S)}{\text{fun}(x)\{e\}, \Sigma \to \langle \Gamma, x, e \rangle, \Sigma} \ (\textsc{LFun})$$

$$\frac{\begin{array}{c} e_1, \Sigma_1 \to v_1, \Sigma_2 \qquad e_2, \Sigma_2 \to v_2, \Sigma_3 \\ v_3 = Op(v_1, v_2) \end{array}}{e_1 \ \text{ op } e_2, \Sigma_1 \to v_3, \Sigma_3} \ (\textsc{LOp})$$

$$\frac{e_1, \Sigma \to \langle \Gamma, x, e_2 \rangle, \Sigma \qquad \langle \Gamma, x, e_2 \rangle, \Sigma \stackrel{D}{\to} a}{\text{toAST}(e_1), \Sigma \to a, \Sigma} \ (\textsc{LAst})$$

$$\frac{\Sigma = (\Gamma, S) \qquad a, \Sigma \stackrel{C}{\to} e}{\text{compile}(a), \Sigma \to e, \Sigma} \ (\textsc{LComp})$$

$$b, \Sigma \stackrel{D}{\to} [\text{base } b] \ (\textsc{DBase})$$

$$\frac{\Sigma = (\Gamma, S, F)}{x, \Sigma \stackrel{D}{\to} [\text{var } \ \langle \Gamma, \_, x \rangle]} \ (\textsc{DVar})$$

$$\frac{e_1, \Sigma \stackrel{D}{\to} a_1 \qquad e_2, \Sigma \stackrel{D}{\to} a_2}{e_1 \ \text{op } e_2, \Sigma \stackrel{D}{\to} [\text{op } a_1 \, a_2]} \ (\textsc{DOp})$$

$$\frac{e, \Sigma \stackrel{D}{\to} a}{\langle \Gamma, x, e \rangle, \Sigma \stackrel{D}{\to} [\text{fn } x \, a]} \ (\textsc{DFn})$$

$$[\text{base } b], \Sigma \stackrel{C}{\to} b, \Sigma \ (\textsc{CBase})$$

$$\frac{\begin{array}{c} \Sigma = (\Gamma, S) \\ e_1, (\Gamma_1, S) \to v_1, \Sigma_2 \end{array}}{[\text{var } \ \langle \Gamma_1, \_, e_1 \rangle], \Sigma \stackrel{C}{\to} v_1} \ (\textsc{CVar})$$

$$\frac{a_1, \Sigma \stackrel{C}{\to} e_1 \qquad a_2, \Sigma \stackrel{C}{\to} e_2}{[\text{op } a_1 \, a_2], \Sigma \stackrel{C}{\to} e_1 \ \text{op } e_2} \ (\textsc{COp})$$

$$\frac{a, \Sigma \stackrel{C}{\to} e}{[\text{fn } x \, a], \Sigma \stackrel{C}{\to} \langle \Gamma, x, e \rangle} \ (\textsc{CFn})$$

Figure 1: Rules $\to$ for the evaluation of Lua expressions, $\stackrel{D}{\to}$ for decompiling Lua expressions into ASTs, and $\stackrel{C}{\to}$ for compiling ASTs back into expressions.

We specify below the behavior of toAST() and compile() by using the formalization of a subset of Lua semantics, as presented (as Lua Core) in [1]. This subset depicts the notions of lexical scoping, closures and side-effects and is therefore sufficient for our purposes. We extend the Lua Core specification from [1] with a general "binary operator" expression, mimicking Lua operators supported by LuaToAST.

Lua Core syntax is presented as follows:

$$e = b \,|\, x \,|\, \text{let } x = e \ \text{ in } e \,|\, x := e \,|\, e(e) \,|\, \text{fun}(x)\{e\} \,|\, e \text{ op } e \,|\, \text{toAST}(e) \,|\, \text{compile}(a)$$

Lua expressions can be base values (b), variables (x), a scoped variable definition (let $x = e$ in e, with $e; e$ as sugar for let $\_ = e$ in e), a variable assignment ($x := e$), an application ($e(e)$), a function definition ($\text{fun}(x)\{e\}$) or an operation on expressions (e op e, with semantics defined by a function Op). We extend this by adding operations toAST(e) and compile(a).

$$v = b \,|\, \langle \Gamma, x, e \rangle \,|\, a$$

Lua values can be base values (b), closures ($\langle \Gamma, x, e \rangle$) or Lua ASTs ($a$).

$$a = [\mathrm{fn}\, x\, a] \,|\, [\mathrm{base}\ b] \,|\, [\mathrm{var}\ \ \langle \Gamma, x, e \rangle] \,|\, [\mathrm{op}\, a\, a]$$

A Lua AST for a function consists of a root node ($[\mathrm{fn}\, x\, a]$) which may contain nodes that wrap base values ($[\mathrm{base}\ b]$), variables ($[\mathrm{var}\ \ \langle \Gamma, x, e \rangle]$) and operations ($[\mathrm{op}\, a\, a]$).

The rules for evaluating Lua expressions over an environment Σ, which is a tuple (Γ, S) containing a namespace $\Gamma : x \to p$ and a store $S : p \to v$ (where p are memory positions) are as follows. We use $\to$ instead of $\xrightarrow{L}$ as in [1]; where rules have the same names, they have the same semantics as those presented in that work.

$$v, \Sigma \to v, \Sigma \ \ (\textsc{LVal})$$

$$\frac{\Sigma = (\Gamma, S)}{x, \Sigma \to S(\Gamma(x)), \Sigma} \ (\textsc{LVar})$$

$$\frac{\begin{array}{c} e_1, \Sigma_1 \to v_1, (\Gamma_2, S_2) \\ p\, \text{fresh} \qquad e_2, (\Gamma_2[x \leftarrow p], S_2[p \leftarrow v_1]) \to v_2, (\Gamma_3, S_3) \end{array}}{\text{let } x = e_1 \ \text{ in}\, e_2, \Sigma_1 \to v_2, (\Gamma_2, S_3)} \ (\textsc{LLet})$$

$$\frac{\begin{array}{c} e_1, \Sigma_1 \to \langle \Gamma_1, x, e_3 \rangle, \Sigma_2 \qquad e_2, \Sigma_2 \to v_1, (\Gamma_3, S_3) \\ p \ \text{ fresh} \qquad e_3, (\Gamma_1[x \leftarrow p], S_3[p \leftarrow v_1]) \to v_2, (\Gamma_4, S_4) \end{array}}{e_1(e_2), \Sigma_1 \to v_2, (\Gamma_3, S_4)} \ (\textsc{LApp})$$

$$\frac{e_1, \Sigma_1 \to v_1, (\Gamma, S) \qquad \Gamma(x) = p}{x := e, \Sigma \to v, (\Gamma, S[p \leftarrow v])} \ (\textsc{LAsn})$$

$$\frac{\Sigma = (\Gamma, S)}{\text{fun}(x)\{e\}, \Sigma \to \langle \Gamma, x, e \rangle, \Sigma} \ (\textsc{LFun})$$

$$\frac{e_1, \Sigma_1 \to v_1, \Sigma_2 \qquad e_2, \Sigma_2 \to v_2, \Sigma_3 \qquad v_3 = Op(v_1, v_2)}{e_1 \ \ \text{op}\, e_2, \Sigma_1 \to v_3, \Sigma_3} \ (\textsc{LOp})$$

$$\frac{e_1, \Sigma \to \langle \Gamma, x, e_2 \rangle, \Sigma \qquad \langle \Gamma, x, e_2 \rangle, \Sigma \xrightarrow{D} a}{\text{toAST}(e_1), \Sigma \to a, \Sigma} \ (\textsc{LAst})$$

$$\frac{\Sigma = (\Gamma, S) \qquad a, \Sigma \xrightarrow{C} e}{\text{compile}(a), \Sigma \to e, \Sigma} \ (\textsc{LComp})$$

The rules for decompiling Lua expressions ($\overset{D}{\rightarrow}$) over an environment Σ are the following:

$$b, \Sigma \overset{D}{\rightarrow} [\text{base } b] \quad (\text{DB}\textsc{ase})$$

$$\frac{\Sigma = (\Gamma, S, F)}{x, \Sigma \overset{D}{\rightarrow} [\text{var } \langle \Gamma, \_, x \rangle]} \quad (\text{DV}\textsc{ar})$$

$$\frac{e_1, \Sigma \overset{D}{\rightarrow} a_1 \qquad e_2, \Sigma \overset{D}{\rightarrow} a_2}{e_1 \text{ op } e_2, \Sigma \overset{D}{\rightarrow} [\text{op}\, a_1\, a_2]} \quad (\text{DO}\textsc{p})$$

$$\frac{e, \Sigma \overset{D}{\rightarrow} a}{\langle \Gamma, x, e \rangle, \Sigma \overset{D}{\rightarrow} [\text{fn } x\, a]} \quad (\text{DF}\textsc{n})$$

Note that $\overset{D}{\rightarrow}$ is defined only for variables, base values and the operator, mirroring the implementation of LuaToAST.

Finally, the rules for compiling Lua ASTs ($\overset{C}{\rightarrow}$) over an environment Σ are:

$$[\text{base } b], \Sigma \overset{C}{\rightarrow} b, \Sigma \quad (\text{CB}\textsc{ase})$$

$$\frac{\Sigma = (\Gamma, S) \qquad e_1, (\Gamma_1, S) \rightarrow v_1, \Sigma_2}{[\text{var } \langle \Gamma_1, \_, e_1 \rangle], \Sigma \overset{C}{\rightarrow} v_1} \quad (\text{CV}\textsc{ar})$$

$$\frac{a_1, \Sigma \overset{C}{\rightarrow} e_1 \qquad a_2, \Sigma \overset{C}{\rightarrow} e_2}{[\text{op}\, a_1\, a_2], \Sigma \overset{C}{\rightarrow} e_1 \text{ op } e_2} \quad (\text{CO}\textsc{p})$$

$$\frac{a, \Sigma \overset{C}{\rightarrow} e}{[\text{fn } x\, a], \Sigma \overset{C}{\rightarrow} \langle \Gamma, x, e \rangle} \quad (\text{CF}\textsc{n})$$

The evaluation of variables x happens only at compilation time, as evidenced by the evaluation of e_1 using $\rightarrow$ in rule CVar.

References

[1] DeVito et al. "Terra: a multi-stage language for high-performance computing" PLDI'13.