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 the 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 ($a$).

$$a = [\text{base } b] \,|\, [\text{closure } \langle \Gamma, x, e \rangle] \,|\, [\text{op } aa]$$

A Lua AST may contain nodes that wrap base values ([*base b*]*)*, closures ([closure $\langle \Gamma, x, e \rangle$]) and operations ([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.

$$v, \Sigma \to v, \Sigma \;\; (\text{LVAL})$$

$$\frac{\Sigma = (\Gamma, S)}{x, \Sigma \to S(\Gamma(x)), \Sigma} \;\; (\text{LVAR})$$

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

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

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

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

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

$$\frac{e_1, \Sigma \to \langle \Gamma, x, e_2 \rangle, \Sigma \qquad e_2, \Sigma \xrightarrow{D} a_1}{\mathrm{toAST}(e_1), \Sigma \to a_1, \Sigma} \ (\text{LAst})$$

$$\frac{\Sigma = (\Gamma, S) \qquad a_1, \Sigma \xrightarrow{C} e_1}{\mathrm{compile}(a_1), \Sigma \to \langle \Gamma, \_, e_1 \rangle, \Sigma} \ (\text{LComp})$$

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

$$b, \Sigma \xrightarrow{D} [\text{base } b] \ (\text{DBase})$$

$$\frac{\Sigma = (\Gamma, S, F)}{x, \Sigma \xrightarrow{D} [\text{closure } \langle \Gamma, \_, x \rangle]} \ (\text{DVar})$$

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

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

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

$$[\text{base } b], \Sigma \xrightarrow{C} b, \Sigma \ (\text{CBase})$$

$$\frac{\Sigma = (\Gamma, S) \qquad e_1, (\Gamma_1, S) \to v_1, \Sigma_2}{[\text{closure } \langle \Gamma_1, \_, e_1 \rangle], \Sigma \xrightarrow{C} v_1} \ (\text{CVar})$$

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

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

References

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