

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 captures the notions of lexing 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:

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

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 ( $e \text{ op } e$ , with semantics defined by the function  $Op$ ). We extend this by adding operations  $\text{toAST}(e)$  and  $\text{compile}(a)$ .

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

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

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

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

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

$$v, \Sigma \rightarrow v, \Sigma \text{ (LVAL)}$$

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

$$\frac{e_1, \Sigma_1 \rightarrow v_1, (\Gamma_2, S_2) \quad p \text{ fresh} \quad e_2, (\Gamma_2[x \leftarrow p], S_2[p \leftarrow v_1]) \rightarrow v_2, (\Gamma_3, S_3)}{\text{let } x = e_1 \text{ in } e_2, \Sigma_1 \rightarrow v_2, (\Gamma_2, S_3)} \text{ (LLET)}$$

$$\frac{e_1, \Sigma_1 \rightarrow \langle \Gamma_1, x, e_3 \rangle, \Sigma_2 \quad e_2, \Sigma_2 \rightarrow v_1, (\Gamma_3, S_3) \quad p \text{ fresh} \quad e_3, (\Gamma_1[x \leftarrow p], S_3[p \leftarrow v_1]) \rightarrow v_2, (\Gamma_4, S_4)}{e_1(e_2), \Sigma_1 \rightarrow v_2, (\Gamma_3, S_4)} \text{ (LAPP)}$$

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

$$\begin{array}{c}
\frac{\Sigma = (\Gamma, S)}{\text{fun}(x)\{e\}, \Sigma \rightarrow \langle \Gamma, x, e \rangle, \Sigma} \text{ (LFUN)} \\
\\
\frac{e_1, \Sigma_1 \rightarrow v_1, \Sigma_2 \quad e_2, \Sigma_2 \rightarrow v_2, \Sigma_3 \quad v_3 = \text{Op}(v_1, v_2)}{e_1 + e_2, \Sigma_1 \rightarrow v_3, \Sigma_3} \text{ (LOP)} \\
\\
\frac{e_1, \Sigma \rightarrow \langle \Gamma, x, e_2 \rangle, \Sigma \quad e_2, \Sigma \xrightarrow{D} a_1}{\text{toAST}(e_1), \Sigma \rightarrow a_1, \Sigma} \text{ (LAST)} \\
\\
\frac{\Sigma = (\Gamma, S) \quad a_1, \Sigma \xrightarrow{C} e_1}{\text{compile}(a_1), \Sigma \rightarrow \langle \Gamma, \_, e_1 \rangle, \Sigma} \text{ (LCOMP)}
\end{array}$$

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

$$\begin{array}{c}
b, \Sigma \xrightarrow{D} [\text{base } b] \text{ (DBASE)} \\
\\
\frac{\Sigma = (\Gamma, S, F)}{x, \Sigma \xrightarrow{D} [\text{closure } \langle \Gamma, \_, x \rangle]} \text{ (DVAR)} \\
\\
\frac{e_1, \Sigma \xrightarrow{D} a_1 \quad e_2, \Sigma \xrightarrow{D} a_2}{e_1 \text{ op } e_2, \Sigma \xrightarrow{D} [\text{op } a_1 a_2]} \text{ (DOP)}
\end{array}$$

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  $\Sigma$  are:

$$\begin{array}{c}
[\text{base } b], \Sigma \xrightarrow{C} b, \Sigma \text{ (CBASE)} \\
\\
\frac{\Sigma = (\Gamma, S) \quad e_1, (\Gamma_1, S) \rightarrow v_1, \Sigma_2}{[\text{closure } \langle \Gamma_1, \_, e_1 \rangle], \Sigma \xrightarrow{C} v_1} \text{ (CVAR)} \\
\\
\frac{a_1, \Sigma \xrightarrow{C} e_1 \quad a_2, \Sigma \xrightarrow{C} e_2}{[\text{op } a_1 a_2], \Sigma \xrightarrow{C} e_1 \text{ op } e_2} \text{ (COP)}
\end{array}$$

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" PLDI13.