Lua & Pallene

Roberto Ierusalimschy

DConf'22
HOW TO: DRAW A HORSE

1. Draw a circle
2. Draw the outline
3. Draw the legs
4. Draw the head
5. Add small details

By: [Author Name]
Lua is a language with lots of small details.
(collaboration with L. H. de Figueiredo and W. Celes)

Pallene is a language still with few details, designed to be used as a system language for Lua in a scripting architecture.
(collaboration with H. Gualandi)
Why Lua?

- Yet another scripting language
- With quite specific goals:
  - Portable
  - Small
  - Simple
  - Emphasis on scripting
Portability

• Runs in virtually any platform with ANSI C.
• Runs inside OS kernels. (e.g., NetBSD)
• Runs directly on the bare metal, without an OS. (e.g., NodeMCU ESP8266)
Size

![Graph showing the growth of different versions of Lua](image-url)

- **Lua 1.0**
- **Lua 4.0**
- **Lua 5.0**
- **Lua 5.1**
- **Lua 5.2**
- **Lua 5.3**
- **Lua 5.4**

**LOC** and **LLOC** are represented by different markers on the graph.
Simplicity

Reference manual with ~100 pages

Documents the language, the C API, and the standard libraries.
Simplicity

- Few but powerful mechanisms
- *Tables*
- Closures
- Coroutines
Tables

- Associative arrays
- Any value as key
- Arrays are tables with integer keys
- Records are tables with literal keys
- Modules are tables, objects are tables
Arrays as Tables

Implementation ensures that tables used as arrays are stored as arrays.

Sparse arrays come for free.
Closures

– First-class anonymous functions with lexical scope
– Widely used in regular Lua code
– Modules are tables populated with anonymous functions
– Exception handling through protected calls
try {
    \texttt{<block/throw>}
}
catch (err) {
    \texttt{<exception code>}
}

\texttt{(Lambda the ultimate block constructor)}

\begin{verbatim}
local ok, err = pcall(function ()
    \texttt{<block/error>}
end)
if not ok then
    \texttt{<exception code>}
end
\end{verbatim}
Coroutines

- Equivalent to cooperative multithreading
- Equivalent to one-shot continuations (call/1cc)
- Covers most uses of full continuations with a fraction of the complexity
Scripting

• Scripting language x dynamic language

• Program written in two languages: a *scripting* language and a *system* language

• System language implements the hard parts: algorithms, data structures

• Scripting language connects those parts
Lua and Scripting

- Lua is implemented as a library.
- Lua has been designed for scripting.
- Good for embedding and extending.
- Embedded in C/C++, Java, Fortan, C#, Perl, Ruby, Python, etc.
Embedded Systems

Samsung (TVs), Cisco (routers), Logitech (keyboards), Volvo (panels), Mercedes (panels), Olivetti (printers), Océ (printers), Ginga (TV middleware), Verison (set–top boxes), Texas Instruments (calculators), Sierra Wireless (M2M), NodeMCU (IoT), Technicolor (gateways), ...
Scripting and Performance

“If it's slow, rewrite it in C”
Easier said than done...

• Data mismatch

• Language mismatch
Data Mismatch

• Data has to be transferred between the two languages.
• Data has to be converted between the two languages.
• This process can kill any gains in performance due to a faster language.
Language Mismatch

- Big differences between Lua and C.
- It can be expensive to convert code from Lua to C.
- It can be hard to predict whether it is worth converting.
What about JITs?

- Don't change the language.
- Can achieve quite good performance.
- Hard to implement, port, and maintain.
- Optimization killers.

Programmers can go to great lengths to appease a JIT. At what point does it become a good idea to switch to a typed language?
Pallene: a Companion Language

- Pallene has been designed to act as a system counter–part of Lua in a scripting architecture.
- To reduce language mismatch, it is a typed subset of Lua.
- To reduce data mismatch, it operates directly on Lua data.
- Scripting + gradual typing.
local function addqueen (N:integer, a:{integer}, i:integer)
  if i > N then
    printsolution(N, a)
  else
    for c = 1, N do
      if isplaceok(a, i, c) then
        a[i] = c
        addqueen(N, a, i + 1)
      end
    end
  end
end
end
Some Design Principles

• Same selling points from Lua.
• Very simple type system.
• Good on the borders.
• Gradual guarantee.
• Simple AOT compiler.
Selling points

Portable, Small size, Simple, Emphasis on scripting
Simple type system

The primary goal of the type system is to help the compiler, in particular to allow *unboxing*!

Everything else can be handled with any, the dynamic type. Or, better yet, kept in Lua.
Good on the borders

Real programs are seldom fully translated. Only the performance-critical parts need optimizations.

The change of one single function from Lua to Pallene should not worsen the performance.
Gradual Guarantee

Pallene functions should have the same semantics of their translation to Lua (by removing type annotations), except for type errors.

Hard to ensure to the letter in a real language like Lua. Simple solution is to remove features from Pallene, which makes it less expressive.
Simple AOT compiler

Generates C code that can be loaded by the standard Lua interpreter, as a C module.

Keeps the same standards of portability as Lua itself.

Simplifies the implementation.
(a) Spectral Norm

(b) Queens

(c) Nbody

(d) Stream Sieve
Conclusions

- Lua is a mature language with strong niches in games and embedded software.
- Lua selling points are simplicity, portability, small size, and emphasis on scripting.
- Performance is an always–present concern for dynamic languages.
- A companion language is an approach for improving the performance of Lua that seems compatible with its selling points.