HsLua

Haskell-bindings to Lua via the C FFI

Albert Krewinkel

Berlin HUG, 2017-08-23

Lua

Small scripting language

Lua is

- well-designed,
- it provides a nice C API,
- embeddable with a small footprint, *and*
- the default when extensibility is desired.

Example programs using Lua

- Redis, Apache, lighttpd, nginx, varnish, prosody,
- many, many games and game engines (e.g., Angry Birds, Civilization VI, SimCity 4, the Witcher, World of Warcraft),
- NetBSD, Damn Small Linux,
- VLC, mpv, awesome, celestia, darktable, WeeChat,
- wireshark, nmap, snort, flame, *and*
- pandoc.

Obligatory Fibonacci example

```lua
function fib(n)
  local a, b = 0, 1
  for i = 0, (n - 1) do
    a, b = b, a + b
  end
  return a
end
function map(fn, tbl)
  local new = {}
  for k, v in pairs(tbl) do new[k] = fn(v) end
  return new
end
print(table.concat(map(fib, {2, 3, 5, 7}), " "))
-- 1 2 5 13
```

HsLua

Haskell bindings

```
main = runLua $ do
  openlibs
  getglobal "print"
  pushstring "Hello from"
  getglobal "_VERSION"
  call 2 0

→ Hello from Lua 5.3
```

Data exchange

Pushing a tuple

```
push ("Hello", True, [40..42])
```

will result in a lua table

```
{"Hello", true, {40, 41, 42}}
```

Calling functions

```
-- define a function in lua:
function greet(greeting)
  return greeting .. " " .. os.getenv("USER")
end

--                fnName  argument
user <- callFunc "greet" "Hello"

-- output: Hello Albert
```

History

Progress happens by scratching an itch:

- Initially developed by Gracjan Polak.
- Improved and maintained by Ömer Sinan Ağacan.
- Now maintained by that guy in front.

Excursus: Working with C

Foreign Function Interface

```haskell
-- Define new function lua_pushinteger
foreign import ccall "lua.h lua_pushinteger"
  lua_pushinteger :: LuaState -> LuaInteger -> IO ()

-- call lua_pushinteger as haskell function
lua_pushinteger l i
```

C data types

Excellent support for working with pointers and the usual C data types:

- pointer data types `Ptr a`, `FunPtr a`, type casting:

  ```
  castPtr :: Ptr a -> Ptr b
  ```

- `int` → `CInt`,
- `double` → `CDouble`,
- `char*` → `CString` (type alias for `Ptr CChar`),
- conversion of strings via

  ```
  withCString :: String -> (CString -> IO a) -> IO a
  ```

- *etc.*

`newtype` everywhere

Newtypes can be used

- to mimic `typedef` definitions;
- in FFI declarations:

```
foreign import ccall "lua.h lua_tointeger"
  lua_tointeger :: LuaState        -- Ptr ()
                -> StackIndex      -- CInt
                -> IO LuaInteger   -- CInt
```

Typing exampels

```c
/* In C */
typedef int (*lua_CFunction) (lua_State *L);
```

```haskell
-- equivalent in Haskell
type CFunction = FunPtr (LuaState -> IO CInt)

newtype StackIndex = StackIndex { fromStackIndex :: CInt }
  deriving (Enum, Eq, Num, Ord, Show)
```

Binding to the Lua C API

Basic example

```
foreign import ccall "lua.h lua_tointeger"
  lua_tointeger :: LuaState -> StackIndex -> IO LuaInteger
```

Cheap optimization with `unsafe`

Functions not calling back into Haskell can be marked `unsafe`.

```
--              Improves performance
--                  considerably
--                       |
foreign import ccall unsafe "lua.h lua_tointeger"
  lua_tointeger :: LuaState -> StackIndex -> IO LuaInteger
```

Cheap optimization with `unsafe`

Functions not calling back into Haskell can be marked `unsafe`.

```
--              Improves performance
--                  considerably
--                       |
foreign import ccall unsafe "lua.h lua_tointeger"
  lua_tointeger :: LuaState -> StackIndex -> IO LuaInteger
```

Has the potential to cause bugs due to GC and finalizers.

Challenges

- Both, Lua and Haskell, have garbage collectors:
  → everything must be copied, especially strings.
- Supported Lua versions differ in their C API:
  → wrappers and CPP directives.
- Error handling with `setjmp`, `longjmp` plays poorly with RTS:
  → C wrappers must be used for error handling.
- Coroutines work via `setjmp` / `longjmp`, which is problematic:
  → currently unsupported, better solution yet to be implemented.

Wrapping up

Summary

- Haskell's FFI allows calling C.
- Newtypes are awesome.
- Lua is great to make your program extensible.
- HsLua makes Lua useable with Haskell.

Further reading

- Main repo: https://github.com/osa1/hslua
- GitHub organisation: https://github.com/hslua
- Project hslua-aeson: push and receive JSON-serializable data to and from Lua;
- Project hslua-examples: example code.