minimalistic typed Lua is here

Hisham Muhammad
<hisham@konghq.com>
minimalism versus types

Hisham Muhammad
<hisham@konghq.com>
minimalistic

experimental

emerging
✅ minimalistic
✅ experimental
✅ emerging
untyped: no types at all
assembly, un(i)typed lambda calculus

typed: types exist!
string and number are different things
(even if you can do "1" + 2)
dynamically typed:
values have types, variables don't
Lua, Scheme, JavaScript, Python, Ruby, PHP, etc.

statically typed:
values have types, variables have types
C, Java, Go, C#, Rust, Haskell, etc.
Python → mypy, pytype
Ruby → Sorbet
PHP → Hack
JavaScript → TypeScript
Racket → Typed Racket
etc.
Lua?
adding types (or anything!) makes a language larger

- conceptually
- and in implementation
adding types (or anything!) makes a language larger

--conceptually
- and in implementation
adding types (or anything!) makes a language larger

--conceptually
- and in implementation
if the language grows too much, it doesn't feel like Lua anymore
if the language grows too much, it doesn't feel like Lua anymore.

if the type checker is too simplistic, it doesn't feel like Lua anymore.
but we want both:

a small language that fits in your head

a type checker that catches when you make a silly typo
the challenge: to find the sweet spot between minimalism and functionality
tl
minimal implementation in the Lua spirit:

Lua: 297 kB tarball
pure standard C, no dependencies

tl: single file, currently 4806 lines
pure Lua, no dependencies
- lexer
- lexer pretty-printer
- parser
- AST traversal
- AST pretty-printer
- type checker
- standard library types
- loader
no dependencies:
drop tl.lua in your Lua project
and off you go
tl check file.tl →

tl gen file.tl → file.lua

tl run file.tl
two modes:

.tl ("strict" mode)

.lua ("lax" mode)
function f(x)
    return x
end

local z = f(0)
function f(x: number): number
    return x
end

local z = f(0)
tl reports
errors and unknowns
separately
type checker: the bulk of the compiler
function keys(t: {string: string}): {string}
    local ks = {}
    for k, v in pairs(t) do
        table.insert(ks, k)
    end
    return ks
end
types of tables
what is a Lua table?
tables in tl:
tables in tl:

maps, like `{string: boolean}`
tables in tl:

maps, like `{string: boolean}`
array, like `{string}`
tables in tl:

maps, like `{string: boolean}`
array, like `{string}`
record, like `Point`
tables in tl:

maps, like `{string: boolean}`
array, like `{string}`
record, like Point
array-record, like Node
tables in tl:

maps, like `{string: boolean}`
array, like `{string}`
record, like Point
array-record, like Node
array-map? not yet
nominal records

Point = record
  x: number
  y: number
end
no inheritance or interfaces/traits (for now?)
with dynamic types, it's trivial to write very generic code
function keys(t: {`K: `V}): {`K}
  local ks = {}
  for k, v in pairs(t) do
    table.insert(ks, k)
  end
  return ks
end
prioritizing practical needs over a feature checklist
yay, types! now what?
which errors are left?
oops.lua:279: attempt to index a nil value (field '?')
stack traceback:
  oops.lua:279: in function 'oh_no'
oops.lua:12: in function 'not_again'
oops.lua:490: in function 'main'
[C]: in ?
tl (and Lua): any variable may be nil
option types?

Maybe in Haskell, Result in Rust, etc...
trickier for Lua:

every $t[x]$ returns an option type? nah
...have the compiler detect it?
dug out of the rabbit hole!
...by the FOSDEM deadline and by user feedback!
<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Union types?</td>
<td>#40 opened 2 days ago by pdesaulniers</td>
</tr>
<tr>
<td>2</td>
<td>Method definition on record imported from declaration file does not</td>
<td>#39 opened 2 days ago by pdesaulniers</td>
</tr>
<tr>
<td></td>
<td>throw an error?</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function overloading in record definitions</td>
<td>#36 opened 3 days ago by pdesaulniers</td>
</tr>
<tr>
<td>2</td>
<td>How to load declaration files that do not correspond to Lua modules</td>
<td>#35 opened 3 days ago by hishamhm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Convenient way of generating Lua files for every .tl file?</td>
<td>#31 opened 3 days ago by pdesaulniers</td>
</tr>
<tr>
<td>2</td>
<td>missing support for exported types</td>
<td>#29 opened 4 days ago by hishamhm</td>
</tr>
<tr>
<td>1</td>
<td>name idea(s)</td>
<td>#25 opened on Nov 24, 2019 by akavel</td>
</tr>
<tr>
<td>1</td>
<td>Would be nice to have more info about the project and it's goals</td>
<td>#24 opened on Nov 22, 2019 by ryanford-frontend</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in Lua mode, warn on assignment of literal with extra fields to a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>record type</td>
<td></td>
</tr>
</tbody>
</table>
Adding types to existing Lua modules & global variables? #28

pdesaulniies commented 4 days ago

Will tl support something equivalent to TypeScript's declaration files?

In LOVE, all of the API is exposed through a global `love` table. I would like to declare all the functions in this table so that I can call them in a type-safe manner.

I would like to do the same for existing Lua libraries as well (such as Penlight).

hishamhm commented 4 days ago

Yes, I have thought about adding that at some point! This might just give me the push to prioritize this. :)

hishamhm commented 4 days ago

@pdesaulniies how about #30 for an initial implementation of this?

Please note that for now all types need to be `global` so they can be used across modules (a limitation noted in #29 which I should fix soon).
definition files

require("socket")

when typechecking, load socket.d.tl
when running, load socket.lua
Some functions in LÖVE have multiple overloads. For instance, `love.graphics.print`.

Right now, it seems like `tl` only checks the last overload:

```plaintext
global love_graphics = record
    print: function(text: string, x: number, y: number, r: number, sx: number, sy: number)
    end

    print: function(coloredtext: {any}, x: number, y: number, r: number, sx: number, sy: number)
end

global love = record
    graphics: love_graphics
end

require("love")

function love.draw()
    love.graphics.print("Hello lol", 100, 100)
end
```

```
main.tl:4:22: argument 1: got string "Hello lol", expected {any}
```
Lua has no function overloading!
but it's common to fake it
challenge:

love.graphics.print({{1,1,1,1}, "Hello", {1,0,0,1}, " World"})
SYNOPSIS

love.graphics.print( colortext, x, y, angle, sx, sy, ox, oy, kx, ky )

ARGUMENTS

`table colortext`
A table containing colors and strings to add to the object, in the form of `{color1, string1, color2, string2, ...}`.

`table color1`
A table containing red, green, blue, and optional alpha components to use as a color for the next string in the table, in the form of `{red, green, blue, alpha}`.

`string string1`
A string of text which has a color specified by the previous color.

`table color2`
A table containing red, green, blue, and optional alpha components to use as a color for the next string in the table, in the form of `{red, green, blue, alpha}`.

`string string2`
A string of text which has a color specified by the previous color.

`tables and strings ...`
Additional colors and strings.

`number x (0)`
The position of the text on the x-axis.

`number y (0)`
The position of the text on the y-axis.

`number angle (0)`
The orientation of the text in radians.

`number sx (1)`
Scale factor on the x-axis.

`number sy (sx)`
Scale factor on the y-axis.

`number ox (0)`
Origin offset on the x-axis.

`number oy (0)`
Origin offset on the y-axis.

`number kx (0)`
Shearing / skew factor on the x-axis.

`number ky (0)`
Shearing / skew factor on the y-axis.
what is the type of colored text?
what is the type of colored text?

1. any
what is the type of colored text?

1. any
2. table
what is the type of colored text?

1. any
2. table
3. {any}
what is the type of colored text?

1. any
2. table
3. {any}
4. {string or {number}}
what is the type of colored text?

1. any
2. table
3. {any}
4. {string or {number}}
5. {[i%2==1]:{number},[i%2==0]:string}
what is the type of coloredtext?

1. any
2. table
3. {any}
4. {string or {number}}
5. {[i%2==1]:{number},[i%2==0]:string}
6. {[i%2==1]:({number}|len==4), [i%2==0]:string}
what is the type of colored text?

1. any
2. table
3. {any}
4. {string or {number}}
5. {[i%2==1]:{number},[i%2==0]:string}
6. {[i%2==1]:([{number}|len==4), [i%2==0]:string}
7. ({{[i%2==1]:([{number}|len==4), [i%2==0]:string}|len%2==0)
what is the type of coloredtext?

1. any
2. table
3. {any}
4. {string or {number}}
5. {[i%2==1]:{number},[i%2==0]:string}
6. {[i%2==1]:({number}|len==4),
   [i%2==0]:string}
7. ({[i%2==1]:({number}|len==4),
   [i%2==0]:string}|len%2==0)
8. ({[i%2==1]:({[0-1]}|len==4),
   [i%2==0]:string}|len%2==0)
what is the type of colored text?

1. any
2. table
3. {any}
4. {string or {number}}
5. {[[i%2==1]: {number}, [i%2==0]: string]}
6. {[i%2==1]: ({{number} | len==4),
   [i%2==0]: string]}
7. ({{[i%2==1]: ({{number} | len==4),
   [i%2==0]: string} | len%2==0})
8. ({{[i%2==1]: ({{[0-1]} | len==4),
   [i%2==0]: string} | len%2==0})
local ColorText = record
    r: number
    g: number
    b: number
    a: number
    text: string
end

function my_typed_print(colortext: {ColorText})
    -- ...
end

my_typed_print(
    {
        {r = 1, g = 1, b = 1, a = 1, text = "Hello"},
        {r = 1, g = 0, b = 0, a = 0, text = " World"}
    }
)
types in Lua — did they deliver?

is it easier to maintain an application?
types in Lua — did they deliver?

is it easier to maintain an application?

YES!
so, in closing
http://github.com/hishamhm tl

release 0.1.0

luarocks install tl

(still looking for a better name!)
Lua and types: join us!

thank you