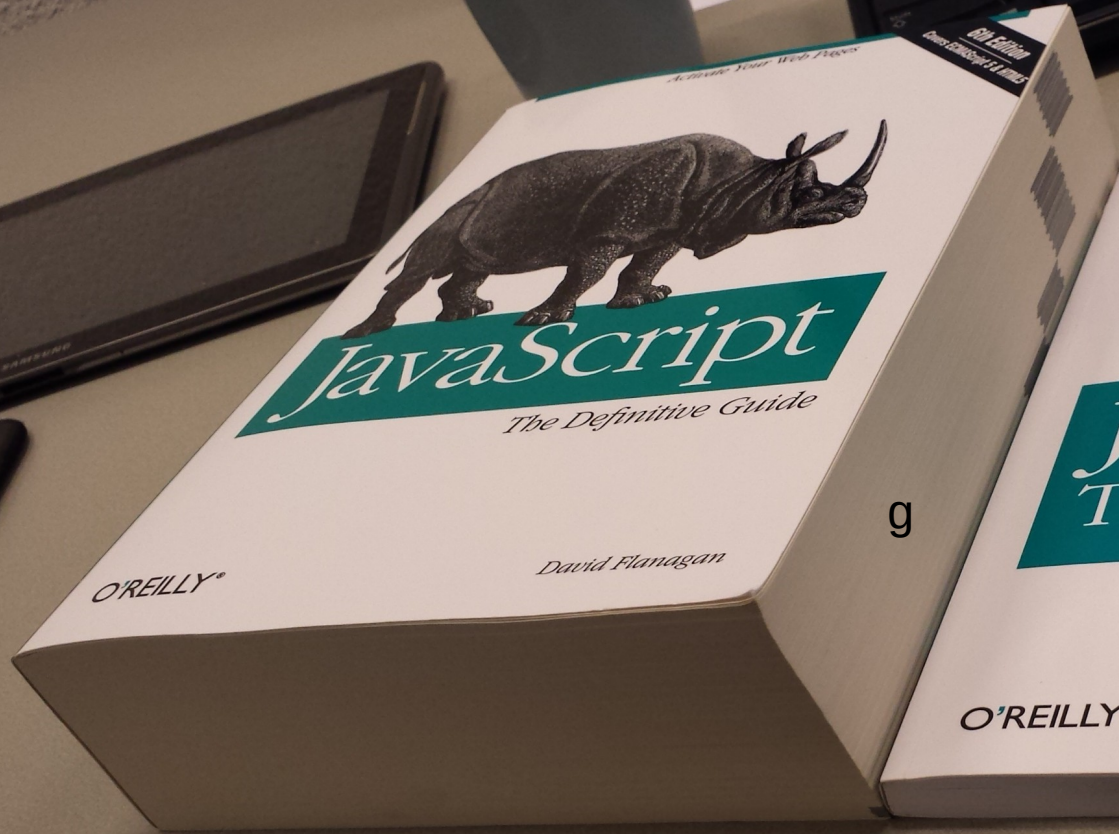


minimalism versus types

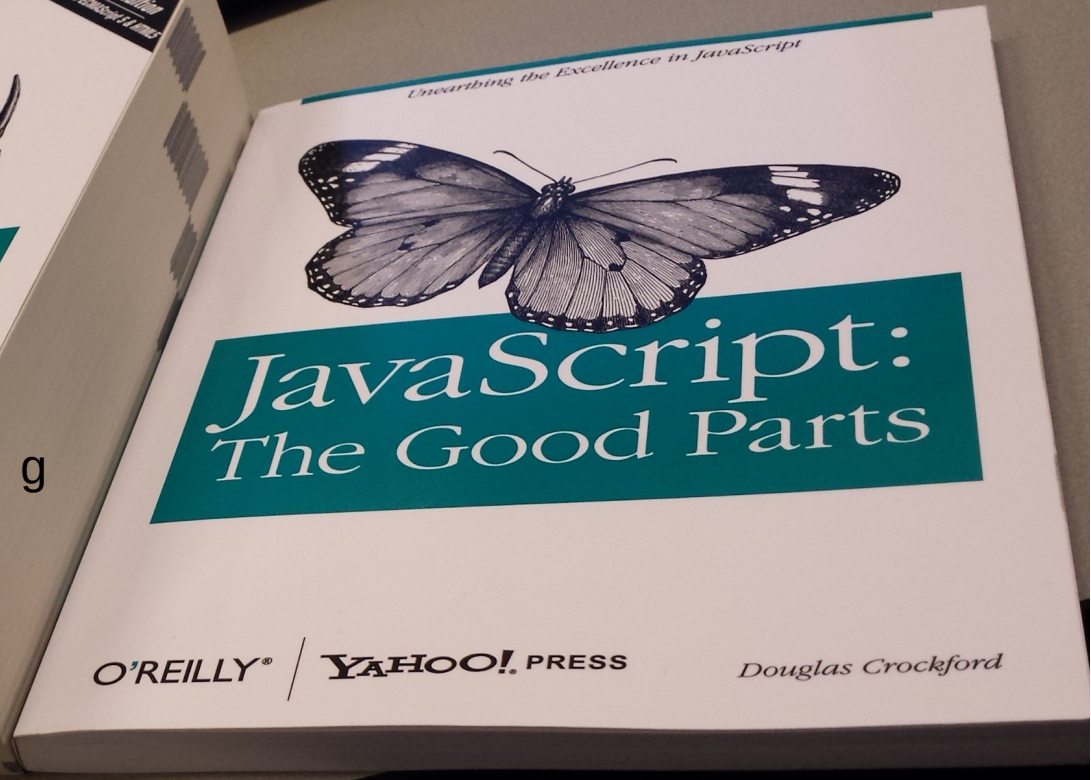
Hisham Muhammad
<hisham@konghq.com>

minimalism

Lua: fits in my head



g



types

pair-programming with the computer

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, Erlang, Python, Ruby, PHP, etc.

statically typed:
values have types, variables have types
C, Java, Go, C#, Rust, Haskell, etc.

~~strongly typed~~

~~weakly typed~~

dynamically typed:
values have types, variables don't
Lua, Scheme, Erlang, Python, Ruby, PHP, etc.

statically typed:
values have types, variables have types
C, Java, Go, C#, Rust, Haskell, etc.

**what happens when we put
minimalism and types together?**

a brief history of typing Lua

2013: Tidal Lock

<https://github.com/fab13n/metalua/blob/tilo/src/tilo/readme.md>

2015: Typed Lua

<https://github.com/andremm/typedlua>

2017: Titan

<https://github.com/titan-lang/titan>

2018: Pallene

<https://github.com/pallene-lang/pallene>

and yet

why is it so hard?

**once you add types and the whole enchilada
that comes with it, then the language is
no longer minimalistic,
right?**

**once you add types and the whole enchilada
that comes with it, then the language is
no longer minimalistic,
right?**

types make our tiny languages complicated?

the problem is kinda the opposite

**dynamically typed languages
have HUGE type systems**

**type system: set of rules that describe
what are the valid interactions of values
in correct programs**

**type system: set of rules that describe
what are the valid interactions of values
in correct programs**

it's in your head!

you are the type checker

```
obj.x, obj.y = get_coords()
```

can't do that:

yesterday I changed `get_coords` **from**

```
return x, y
```

to

```
return { x = x, y = y }
```


what are the rules in your head?

dynamically typed:

values have types, variables don't

Lua, Scheme, Erlang, Python, Ruby, PHP, etc.

statically typed:

values have types, variables have types

C, Java, Go, C#, Rust, Haskell, etc.

dependently typed:

values have types, variables have types...

and types have values! and types!

Idris, Agda... not that many yet!

function f(a, b)

```
function f(a, b)
```

a: integer

b: if a < 256 then string else array of strings

function f(a, b)

a: integer

b: if a < 256 then string else array of strings

red: integer
green: integer
blue: integer

```
function f(a, b)
```

a: integer

b: if a < 256 then string else array of strings

```
    red: integer
```

```
    green: integer
```

```
    blue: integer
```

```
f(red, {1, 2, 3})
```

Curry-Howard correspondence

propositions \leftrightarrow types

logic \leftrightarrow type system

btw, arithmetics is undecidable


```
local t = {}
local ok, err = load_values_into_table(t)
if not ok then
    return nil, "failed! " .. err
end
return { r = t[1], g = t[2], b = t[3] }
```

Lua: table is the only structured type

everything is a table

a table is anything

everything is a table

**a table is anything
an array**

everything is a table

a table is anything

an array

a dictionary

everything is a table

a table is anything

an array

a dictionary

a struct

everything is a table

a table is anything

an array

a dictionary

a struct

an object

everything is a table

a table is anything

an array

a dictionary

a struct

an object

**a dictionary mapping objects to strings or
arrays depending on whether field x of the
key object is true or false**

expressiveness

not really ~~what~~ a language can express

but how can you express it

**dynamically typed languages are
super expressive**

like a blank sheet of paper

type checker works both for good
("Thank you for catching my silly typo!")

and bad

("no, I _know_ that this use of the variable is safe!")

expressiveness is the feel of a language

```
local t = {}  
t.name = "items"  
t[1] = 100  
t[2] = 200
```

here's the dilemma:

how much of the language do you change?

**if you want to make it feel like Lua,
then the type checker is super complex**

**if you want to finish your type checker,
you have to make cuts somewhere**

two options on where to make cuts

cut on programmer expressiveness

```
{ name = "items", items = {100, 200} }
```

```
return x, y and return nil, err
```

vs.

```
return x, y and return nil, nil, err
```

cut on the correctness of the type checker

~~"every program the type checker accepts has
correct types"~~

"every program that the type checker rejects
has wrong types"

**the more sophisticated your type system,
the deeper you are in research territory**

soundness vs. usability (vs. performance!)

Typed Lua
Typed Clojure
Typed Racket

is all lost?

TypeScript

usability above all else

<https://github.com/Microsoft/TypeScript/wiki/TypeScript-Design-Goals>

intentionally unsound

🚩 1 Open ✓ **570 Closed**

Author ▾








Labels ▾

Projects ▾

Milestones ▾

Assignee ▾

Sort ▾

- 🚩 **No error for invalid module augmentation in '.d.ts' file** **By Design**  2
#24307 by andy-ms was closed on May 22, 2018
- 🚩 **Watch mode sometimes watches files that aren't targeted for compilation, depending on installed packages** **By Design**  7
#23414 by dguo was closed on Apr 28, 2018  TypeScript 2.9 
- 🚩 **Distribute conditional types over concrete unions** **By Design**  3
#22595 by DanielRosenwasser was closed on Mar 29, 2018
- 🚩 **isValidSpreadType doesn't handle type parameter** **By Design**  1
#20013 by andy-ms was closed on Nov 14, 2017
- 🚩 **Function that always throws is not inferred as having `never` return type** **By Design**  5
#16608 by masaeedu was closed on Aug 17, 2017
- 🚩 **Keyof doesn't see all apparent members** **By Design**  3
#16578 by olegdunkan was closed on Jun 17, 2017

what about Lua?

exploring this design space

tl: minimalistic Lua type checker

**what's the minimum set of features
so that it can check itself?**

tl tl.tl: currently fails with 384 type errors

**tl tl.tl: currently fails with 384 type errors
(one week ago it was 1493!)**

TypeScript: JavaScript-like
(features, features, features!)

tl: aiming for Lua-like
(a balance between functionality and small size)

<http://github.com/hishamhm/tl>

so, in closing

Lua and types: to be continued!

thank you