Group Therapy for the Type-Curious

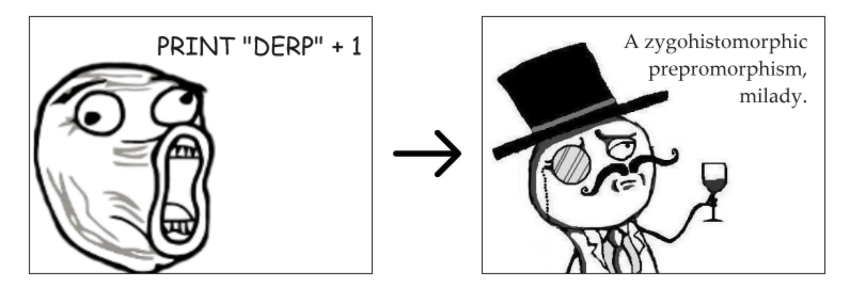
Theory & Practice

Bruce C. Miller

bm3719@gmail.com

November 14, 2018

Our Noble Journey



Overview

Type Theory Introduction and fundamentals

Type Systems Type theory applied to programming languages

Static vs. Dynamic And other dichotomies

Types vs. Clojure Begun, the Type Wars have

History of Type Theory

Some History:

- 1902 Types are first proposed by Bertrand Russell as a solution to Russell's Paradox in Cantor's naïve set theory.
- 1940 Types are first applied to programming language theory, combined with Alonzo Church's λ-calculus.
- 1972 System F created. Later to influence ML, Caml, Haskell.
- 1972 Per Martin-Löf's intuitionistic type theory introduced. Creates what's now known as dependent type theory, as used in Agda, Idris, Coq, Lean.
- 2009 What is now known as homotopy type theory introduced in a paper by Voevodsky.

Type Judgments

An introduction to type theory, just as relates to type systems.

Judgments describe type systems.

$$\Gamma \vdash \Im$$

$\Im$ is an assertion. Γ here is the static typing environment, and could be the empty set $\emptyset$, or a list of variables and their types.

$$\Gamma \vdash M : A$$

M has type A in Γ.

$$\emptyset \vdash \mathit{true} : \mathit{Bool}$$

$$\Gamma \vdash \Diamond$$

Type Rules and Derivations

Judgments compose type rules, which compute type derivations.

General form:

$$\frac{\Gamma_1 \vdash \Im_1 \ldots \Gamma_n \vdash \Im_n}{\Gamma \vdash \Im} \; (\mathit{Rule\ name})(\mathit{annotations})$$

Some examples:

$$\frac{}{\emptyset \vdash \Diamond} \; (\mathit{Env}\ \emptyset)$$

$$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash n : \mathit{Nat}} \; (\mathit{Val}\ n)(n = 0, 1, \ldots)$$

$$\frac{\Gamma \vdash M : \mathit{Nat}, \Gamma \vdash N : \mathit{Nat}}{\Gamma \vdash M + N : \mathit{Nat}} \; (\mathit{Val}\ +)$$

Type Systems

From Pierce [3]:
A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

Some notes:

- Languages without type systems are untyped.
- Type systems are parts of programming languages or can be implemented on their own.
- The line between the type system and general language implementation can be fuzzy, e.g. duck typing, type coercion.
- Type safety is the degree to with a language enforces the prevention of type errors.

Type Dichotomies

typed	untyped
static	dynamic
explicit manifest	implicit inferred
structural	nominal
sound	unsound
intrinsic	extrinsic

There are some conversational dichotomies that don't really have firm definitions:

weak	strong
loose	tight?

The Static Typing Position

Aggregated from Haskell user opinions:

- **Reduction in bugs** Detection at compile time. Closed vs. open programs.
- **Communicating intent** Type signatures, type declarations, etc.
- **Encoding program logic** Making bugs into type errors, refactoring.
- **Easier for n00bs** Naming nouns, nouns describe what they are/do.
- **Esoterica** Type-level programming, C-H, CT, etc.

The Dynamic Typing Position

Aggregated from Clojure user opinions:

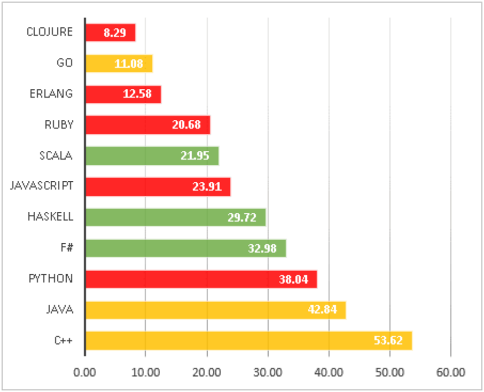
- **Real world data** In real life, data is sparse and/or unstructured.
- **Syntactic overhead** Code density, reduced logic distribution, small-scale development, REPLs.
- **Ease of Maintenance** Adding to and refactoring code.
- **Lazy-loading needed parts** spec, core.typed, core.logic
- **Type errors vs. real life** Clojure ranks as one of the lowest bug-prone languages.

Resources

- *Group Therapy for the Type-Curious*, `https://github.com/bm3719/types-talk`.
- R. Nedepelt, H. Geuvers, *Type Theory and Formal Proof: An Introduction*, 1st Edition, Cambridge Univ. Press, Cambridge, MA, 2014.
- B. Pierce, *Types and Programming Languages*, MIT Press, Cambridge, MA, 2002.
- J. Girard, *Proofs and Types*, Cambridge Tracts on Comp. Sci., Cambridge, MA, 1989.
- B. Pierce, *Advanced Topics in Types and Programming Languages*, MIT Press, Cambridge, MA, 2004.

Languages compared

Languages sorted by bug density (100+ star repos):



https://dev.to/danlebrero/the-broken-promise-of-static-typing

The Rosetta Stone

Category Theory	Physics	Topology	Logic	Computation
object X	Hilbert space X	manifold X	proposition X	data type X
morphism $f\colon X \to Y$	operator $f\colon X \to Y$	cobordism $f\colon X \to Y$	proof $f\colon X \to Y$	program $f\colon X \to Y$
tensor product of objects: $X \otimes Y$	Hilbert space of joint system: $X \otimes Y$	disjoint union of manifolds: $X \otimes Y$	conjunction of propositions: $X \otimes Y$	product of data types: $X \otimes Y$

The Rosetta Stone: Homotopy Types

Types	Logic	Sets	Homotopy
A	proposition	set	space
$a{:}A$	proof	element	point
$B(x)$	predicate	family of sets	fibration
$b(x){:}B(x)$	conditional proof	family of elements	section
0,1	$\bot, \top$	$\emptyset, \{\emptyset\}$	$\emptyset, *$
$A + B$	$A \vee B$	disjoint union	coproduct
$A \times B$	$A \wedge B$	set of pairs	product space
$A \to B$	$A \Rightarrow B$	set of functions	function space
$\sum_{(x:A)} B(x)$	$\exists_{x:A} B(x$	disjoint sum	total space
$\prod_{(x:A)} B(x)$	$\forall_{x:A} B(x)$	product	space of sections
id_A	equality $=$	$\{(x,x) \vert x \in A\}$	path space A^I