Comparing: Haskell, Scala, Go

Allele Dev (@queertypes)

August 28, 2014

Contact Me

- Github: cabrera
- Twitter: @queertypes
- Blog: Read, Review, Refactor

Overview

- Intent
- Language summaries
- Library ecosystem
- Tools
- Type systems
- Known issues
- Learning resources
- Recommendations

Intent

- To summarize options under consideration

Intent

- To summarize options under consideration
- To explore ecosystems and tooling for options

Intent

- To summarize options under consideration
- To explore ecosystems and tooling for options
- To compare the type system level guarantees and ability to abstract/reuse

Intent

- To summarize options under consideration
- To explore ecosystems and tooling for options
- To compare the type system level guarantees and ability to abstract/reuse
- To document known issues

Intent

- To summarize options under consideration
- To explore ecosystems and tooling for options
- To compare the type system level guarantees and ability to abstract/reuse
- To document known issues
- To point to resources to learn more

Intent

- To summarize options under consideration
- To explore ecosystems and tooling for options
- To compare the type system level guarantees and ability to abstract/reuse
- To document known issues
- To point to resources to learn more
- To offer strategic recommendations

Haskell

- Appeared: 1990
- Home page: current, WIP
- Compiler: GHC
- Latest release: 7.8.3, July 11, 2014
- Native assembly generation
- OS: Linux, Windows, OS X >=10.7, iOS, FreeBSD, Solaris
- Platforms: x86, ARM
- Paradigms: functional, non-strict
- Notes: Renowned type system, concurrent/fast runtime, cryptol
- Try: Haskell

Haskell

```haskell
factorial :: Integral a => a -> a
factorial n
  | n < 2 = 1
  | otherwise = n * factorial (n - 1)

data Tree a =
    Empty
  | Branch a (Tree a) (Tree a) deriving (Show, Eq)

insert :: Ord a => Tree a -> a -> Tree a
insert Empty x = Branch x Empty Empty
insert (Branch v l r) x
  | x <= v = Branch v (insert l x) r
  | x > v = Branch v l (insert r x)
```

Scala

- Appeared: 2004
- Home page: site
- Compiler: Scala
- Latest release: 2.11.2, July 24, 2014
- JVM byte code generation
- OS: Anything that can host the JVM
- Platforms: JVM
- Paradigms: functional, object-oriented, strict
- Notes: Effective type system, leverages JVM libraries, spark
- Try: Scala

Scala

```scala
sealed trait Tree[A]
case class Empty[A]() extends Tree[A]
case class Branch[A](v: A, l: Tree[A], r: Tree[A])
  extends Tree[A]

object samples {
  def insert[A <% Ordered[A]]
    (t: Tree[A], x: A): Tree[A] = t match {
    case Empty() => Branch(x, Empty(), Empty())
    case Branch(v, l, r) =>
      if (x <= v) Branch(v, insert(l, x), r)
      else        Branch(v, l, insert(r, x))
  }
  def factorial(n: Int): Int = {
    if (n < 2) 1 else n * factorial (n - 1)
  }
}
```

Go

- Appeared: 2009
- Home page: site
- Compiler: Go
- Latest release: 1.3.1, August 13, 2014
- Native assembly generation
- OS: Linux, OS X, Windows, BSDs
- Paradigms: imperative, object-oriented
- Notes: Concurrency support, fast compilation, docker
- Try: Go

Go

```
func factorial(n int) int {
    if n < 2 {
        return 1
    }
    return n * factorial(n - 1)
}

type Tree struct {
    l, r *Tree
    v interface{}  // not type-safe; think (void *)
}

func insert(t Tree, x interface{}) Tree {
  // not-even-going-to-try.jpg
}
```

At a Glance (compiler, rts, stdlib, tests)

Summary	Haskell	Scala	Go
Appeared	1990	2004	2009
Latest Release Date	July 2014	July 2014	August 2014
Platform	x86, ARM*	JVM	x86
Paradigm	Functional, Imperative	OO, Functional	OO, Imperative
REPL	Yes	Yes	No
LOC Main	394539 (Haskell)	268572 (Scala)	432018 (Go)
LOC Other	45760 (C)	29919 (Java)	151908 (C)

Library Support

- A language without a breadth of a libraries is a language that is rarely used
- A language lacking package management infrastructure is harder to adopt
- An FFI is important to leverage works that came before

Haskell

- Package index: Hackage/Stackage
- Count: >6000
- Package manager: cabal
- Package format: Cabal file - example
- FFI: Yes (C, JS)

Scala

- Package index: Maven
- Count: >80000 (mixed with Java)
- Package manager: sbt, others
- Package format: scala example
- FFI: Yes (Java, JNI/C)

Go

- Package index: Go-Search
- Count: >50000
- Package manager: gopm (experimental)
- Package format: .gopmfile (CONF)
- FFI: Yes (C)

At a Glance

Packages	Haskell	Scala	Go
Index	Hackage	Maven	Go-Search
Count	>6000	>80000 (+Java)	>50000
Manager	cabal	sbt	gopm (exp.)
FFI	C, JS	Java, C	C

Tooling

- What editors are available?
- How about IDEs?
- Profiling?
- Debugging?
- Others?

Haskell

- Editors
  - emacs + ghc-mod
  - vim + ghc-mod
  - EclipseFP
- Profiling
  - GHC
  - criterion
  - ThreadScope
  - Heap Profiler
  - Test Coverage
- Debugging: N/A
- Others
  - Type search: Hoogle, Hayoo
  - Style: hlint

Scala

- Editors
  - emacs + scala-mode2 + ensime
  - vim + vim-scala
  - Eclipse + Scala IDE
- Profiling
  - ScalaMeter
  - Java HeapAudit
- Debugging
  - Scala IDE
- Others
  - Type search: Scalaex
  - Linting: Wart Remover, Scala Style

Go

- Editors
  - emacs + go-mode
  - vim + go-mode
  - Various IDEs, including IntelliJ
- Profiling
  - pprof
  - go testing bench
- Debugger
  - gdb
- Others
  - Linting: govet

Type Systems

- A programming language is a frontend to its type system
- A powerful type system is a proof engine
    - Curry-Howard Isomorphism
- Proofs are the only means to rule out errors; testing cannot do this

At a Glance

Type System			
	Haskell	Scala	Go
Analysis Time	Static	Static	Static
Immutable Default	Yes (all)	No	No
1st-Class Functions	Yes	Yes	No
Type Inference	Yes	Yes*	Poor
Evaluation Model	Lazy	Strict	Strict
Modules	Yes (weak)	Yes (strong)	Yes (strong)

- Scala type inference will sometimes yield an Any

At a Glance

Type System	Haskell	Scala	Go
Implicit Casts	No	Yes	No*
Generics	Yes	Yes	No
Higher Kinds	Yes	Yes	No
Nullable Values	No	Yes	Yes
Strong Type Alias	newtype	case class	No

- There's a case where Go allows for implicit conversion

At a Glance

Type System	Haskell	Scala	Go
Sum Types	Yes	Yes	No
Product Types	Yes	Yes	No
Recursive Types	Yes	Yes	No
Pattern Matching	Yes	Yes	No
Effect Tracking	Yes	Possible*	No

- Effect tracking can be achieved via scalaz, with a few caveats

At a Glance

Type System	Haskell	Scala	Go
Overloading	Typeclass	Implicits	No
Records	Yes	Yes	Yes
Subtyping	No*	Yes	Yes
Dependent Types	No*	No*	No

- Subtyping impedes static analysis
- Dependent types can be faked in type systems on par with Haskell's/Scala's, within limits

Known Issues

- Compilers aren't free of defects
- Adopting a language entails owning these defects and working around quirks

At a Glance

Issues	Haskell	Scala*	Go
Known	942	4772	1216
Critical	5	157	N/A
Major	42	443	N/A
FFI	C, JS	Java, C	C

- Scala issues include: compiler backend, collections, concurrent lib, enumeration, macros, misc. compiler, optimizer, pattern matcher, presentation compiler, quasiquotes, reflection, repl
- Go issue tracker does not support priorities

Haskell

- Tracker
- Notable:
  - int-to-float broken on ARM
  - Cabal Hell (with sandbox workaround): more

Scala

- Tracker
- Notable: Listen to Paul . Phillips
    - tl;dr - issues with type inference, casting, and inheritance
    - tl;dr2 - issues have long turn-around time
    - Runar: more criticisms of Scala
    - Suggestions for improving Scala are abundant

Go

- Tracker
- Notable
  - No support for generics
  - Extensibility issues

Learning Resources

- Picking up a new language takes some effort
- Availability of channels to learn should be considered in choosing a language

Haskell

- Learn You a Haskell for Great Good: site
- Real World Haskell (dated): site, what's outdated?
- Parallel and Concurrent Programming in Haskell: site
- Emacs Integration: site
- Setting up a Project: site
- Many, many research papers: index
- What I Wish I Knew When Learning Haskell: site
- Community Curated Learning Guides: site
- Style Guide: site

Scala

- Programming in Scala: site
- Functional Programming in Scala: buy
- Scala for the Impatient: buy
- Twitter's Scala School: site
- Style Guide: site

Go

- Effective Go: site
- How to Write Go: site
- An Introduction to Programming in Go: site
- Go Bootcamp: site
- Style Guide: site

Recommendations: Go

- An improvement over untyped languages, safety-wise
- Lack of generics harms safety, abstraction, and reuse
- Lack of package index/manager harms adoption
- Feature-starved type system impedes use of modern patterns
- Familiar patterns for OO/imperative programmers available
- Fast compilation time is nice
- My thoughts: the weakest of these three choices
- **Recommendation**: use only to modify an existing (go) code base

Recommendation: Scala

- Potent type system can lead to proofs of correctness in code
- Pro: association with JVM means access to JVM libraries
- Con: association with JVM carries over JVM problems
- Developers can use OO patterns or adopt pure FP
  - Good: familiar, lower barrier to entry, a cleaner Java
  - Bad: mutable state abounds, coupling, need more trust in team setting
- **Recommendation**: great! Use especially if you need access to JVM libraries

Recommendations: Haskell

- Potent type system can lead to proofs of correctness in code
- Requires unlearning old patterns
  - Accelerated greatly by communal knowledge and breadth of resources
  - With prior FP knowledge, takes a few days to get ramped up
- Purity aligns development style more closely in team setting
  - Less room for errors, greater chance of correctness if it compiles
  - Applies to third-party libraries by extension
- Great selection of libraries
- Great for understanding link between proofs, mathematics. and programming
- **Recommendation**: excellent! prefer to Scala if JVM libraries not needed

What I Left Out

- Industry use
- Academic use
- Community events
- Notable libraries (property testing, async, web, etc.)
- Runtime system comparison (GC style, performance, memory, etc.)
- Representative applications
- Other powerful languages:
  - Ocaml, Idris, typed Erlang, typed Racket, Rust, Elm, Purescript

Caveats

- I specialize in Haskell
  - I have less knowledge of Go/Scala tools/resources
- My biases:
  - Preference for non-optional. strong typing
  - Preference for functional programming
  - Preference for purity
- My belief: FP + types -> (working code, sooner) & (easier maintenance)

Thank You!