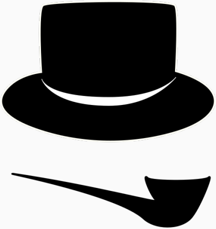
Haskell & Servant

An introduction to the Functionnal Programming paradigm

FlogFR

October 18, 2022



Only my opinion in this talk, trust me

My vision of computer science

- Software is data, and calculation on data (and displaying the results)
- Software is for all kind of industry
- Software development should be as easy/difficult as the problem

My experience

- First love with C++
- Perl as the first language profesionnaly
- Years of experience in Python
- Years of experience in PostgreSQL
- Now full-time Sysadmin/SRE: Good knowledge of what is a production environment
- Never really finished any side project I started. (but I have 1000+ POCs somewhere in the cloud)

My feelings

- The style of coding in all the language I used is not consistent across projects
- Refactoring/Updating the architecture of a medium to big size project (200k+ LOC) is a pain in Python. Mostly because you touch something in one tiny place, and it breaks something at the other side of world.
- The object paradigm is obfuscating the calculation in the code
- I miss having a compiler (like the one for C++)
- Hard to understand the representation of the data in memory in Python

The solution to all your problems

Solution

- Seriously? You thought there's a universal solution to all your problem?

I have a web project

My requirements

- Minimalist architecture and infrastructure to iterate quickly
- No brainer: PostgreSQL + Sqitch
- Give a try to what I heard is "FP"

Let’s test ELM

- https://elm-lang.org/
- Let’s do the official guide (https://guide.elm-lang.org/)
- Let’s create some proof of concept with ELM (ended up to be almost a copy of the SPA example https://github.com/rtfeldman/elm-spa-example)

What I learned from ELM

- pro: Amazing centralized documentation: https://package.elm-lang.org/
- pro: My POC was production ready from day 1 (mostly because of type safety, and the tooling is minimal/simple)
- pro: Finished a small SPA project in 2 months (learning the language included)
- cons: I didn't learn how to unit test because of the transpiler + typing
- cons: The roadmap is not clear of the language
- cons: Frustrated by the language after a time (repetitive)

So I finished one project?

So I finished one project?

- Yes

Let’s analyze this success

- I like the syntax of ELM (subset of haskell)
- I Love the transpiler
- I Love the strong typing

Can I apply now this success to the backend development?

Now Haskell

- Created in 1990, financed by Microsoft (20 years of reseach on the language/compiler)
- *Pure* Functional Programing
- Compiled (marvelous compiler, a masterpiece)
- Lazy (default performance comparable to C programs - be careful of the benchmark online about Rust etc. We had the same with Go couple of years ago)
- Strongly typed - Value level, and the Type level
- Mature and friendly community, stable eco-system. . .
- Close to mathematics

What else? Let’s start to learn it!

Learning Haskell

- Books
- Haskell Wiki
- Community (IRC, Github)
- Github Searches
- Project! Project! Project!

The amazing Haskell

- Haskell documentation search engine ”hoogle”

The amazing Haskell

- Hoogle documentation

```haskell
-- Search: Maybe a -> Bool
-- Results:
isJust :: Maybe a -> Bool
```

The amazing Haskell

- Haskell documentation search engine "hoogle"
- Code makes it straightforward to see the representation of the data and the calculation

The amazing Haskell

The amazing Haskell

- Haskell documentation search engine "hoogle"
- Code makes it straightforward to see the representation of the data and the calculation
- Lazyness = performant code by default

The amazing Haskell

```
-- recursive thinking with performance
```

The amazing Haskell

- Haskell documentation search engine "hoogle"
- Code makes it straightforward to see the representation of the data and the calculation
- Lazyness = performant code by default
- Easy composition of code (thanks to typeclass)

The amazing Haskell

- Generic algorithm

```haskell
class SqlRow a where
  fromSqlResultRows :: Result -> IO [a]
  fromSqlResultRows sqlResult = do
    (Row nbRows) <- ntuples sqlResult
    let rows = [0..(nbRows-1)]
    mapConcurrently (\k -> fromSqlResultRow sqlResult k) rows
```

The amazing Haskell

- Haskell documentation search engine ”hoogle”
- Code makes it straightforward to see the representation of the data and the calculation
- Lazyness = performant code by default
- Easy composition of code (thanks to typeclass)
- Monad (context of calculation) limiting the side effects

The amazing Haskell

- Controlling the side effects

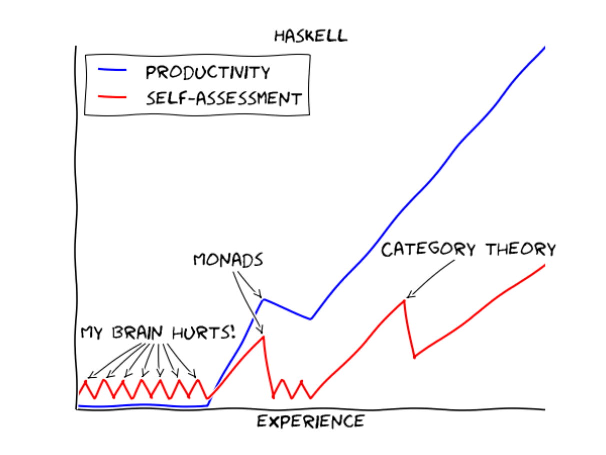
```haskell
-- example 1 (Monad)
type HandlerM = ReaderT SharedEnv (LoggingT Handler)
-- userLoggedInOrRedirect :: Session -> HandlerM ()

-- example 2 (TypeClass)
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```

The amazing Haskell

- Haskell documentation search engine ”hoogle”
- Code makes it straightforward to see the representation of the data and the calculation
- Lazyness = performant code by default
- Easy composition of code (thanks to typeclass)
- Monad (context of calculation) limiting the side effects
- Learning curve and possibilities unlimited (related also to monad)

The amazing Haskell

- Monad/Context of calcultation is also a way of thinking
- Thus you can specialize in any industry



The amazing Haskell

- Haskell documentation search engine ”hoogle”
- Code makes it straightforward to see the representation of the data and the calculation
- Lazyness = performant code by default
- Easy composition of code (thanks to typeclass)
- Monad (context of calculation) limiting the side effects
- Learning curve and possibilities unlimited (related also to monad)
- Easy maintainability and refactoring of code (thank you compiler + strongly type + monad)

The amazing Haskell

```haskell
-- example 1 (Monad)
type HandlerM = ReaderT SharedEnv (LoggingT Handler)
-- userLoggedInOrRedirect :: Session -> HandlerM ()

-- example 2 (TypeClass)
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```

The amazing Haskell

- and I'm forgetting lots of others pros. . .
- cons: I still didn't learn how to do unit tests in Haskell. . .

I want to make an API for my project, in Haskell!

The amazing Servant

- Set of Haskell library to create API
- +1100 github stars
- Describe an API at the Type Level
- Structure re-usable by all libraries

The amazing Servant

- Describe an API at the Type Level

The amazing Servant

```haskell
type FrontAPI =
    -- Public Area
    Header "X-Real-IP" Text :> QueryParam "lat" Double :> ... :> Get '[HTML]
    -- ...
    :<|> "account" :> Get '[HTML] H.Html
    :<|> "account" :> MultipartForm Mem AccountForm :> Post '[HTML] H.Html
```

The amazing Servant

- Describe an API at the Type Level
- Re-usable Type level API structure

The amazing Haskell

```haskell
-- Swagger in one line of code
BSL8.putStrLn $ encode $ toSwagger (Proxy :: Proxy UserAPI)
  & info.title        .~ "User API"
  & info.version      .~ "1.0"
  & info.description  ?~ "This is an API for the Users service"
  & info.license      ?~ "MIT"
  & host              ?~ "example.com"
:}
```

The amazing Servant

- Describe an API at the Type Level
- Re-usable Type level API structure
- Composition of API

The amazing Servant

```
type API =
  "static" :> Raw
  :<|> "blog" :> BlogAPI
  :<|> "login" :> ServerLoginAPI
  :<|> MonitoringAPI
  :<|> AuthProtect "custom-auth" :> FrontAPI
  :<|> AuthProtect "custom-auth" :> "api" :> "v1" :> APIV1
  :<|> AuthProtect "custom-auth" :> "admin" :> AdminAPI
```

Contact

Florian Grignon - @FlogFR
Private Detective @Infratelier

`https://github.com/FlogFR`

`grignon.florian@gmail.com`

Thank You

To my dear friend and associate:

- Dr Watson

A personnal thank you to:

- Organizers and sponsors of the events
- DBAs Colleagues @PeopleDoc
- PostgreSQL community

Questions

FlogFR - grignon.florian@gmail.com - https://github.com/FlogFR