Haskell in Production

Marek Kidoň

20.4.2019

About: Early Days

Back in the early days

- Python
- Perl
- Bash
- C
- Java

I Almost quit programming

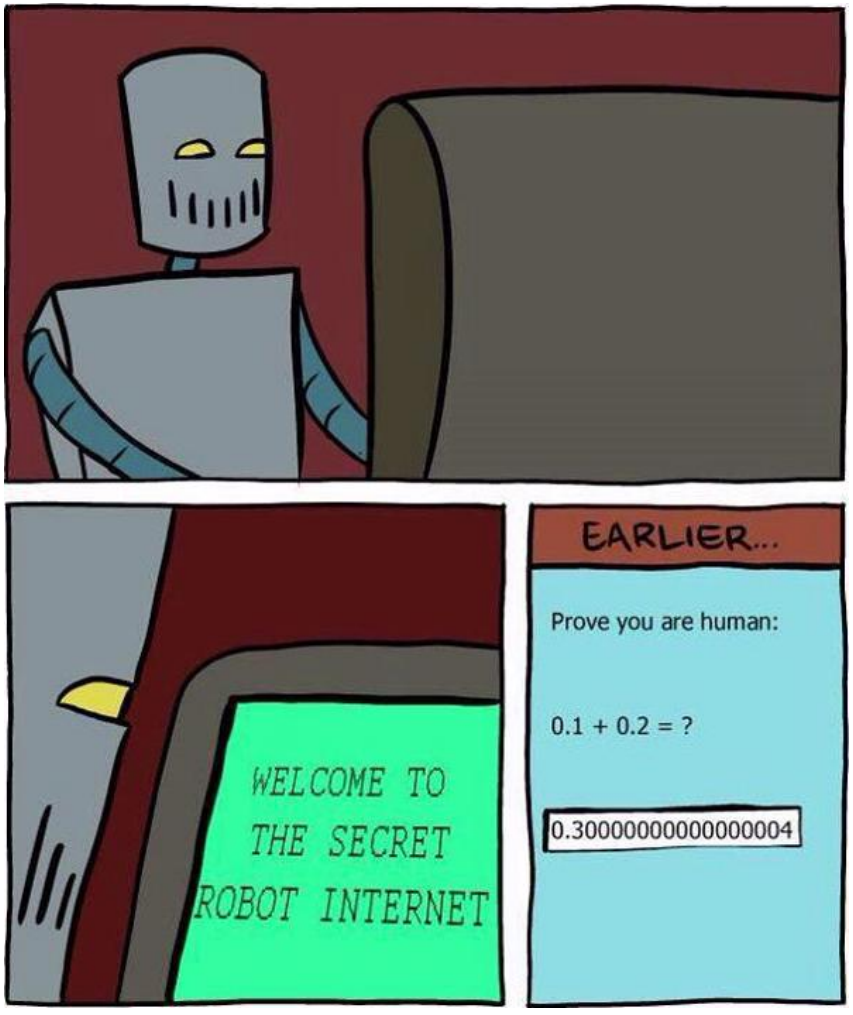
About: Now

- FP Advocate
- 3 years of *Haskell* experience
- 1 year of purely functional *Scala*

Human Machine Correspondence

There is no correspondence between human and machine. Only divergence.

- Machines are superb at calculating things, humans are not.
- Humans are great at symbolic reasoning, machines are not.


WELCOME TO
THE SECRET
ROBOT INTERNET
EARLIER...
Prove you are human:
0.1 + 0.2 = ?
0.30000000000000004

Abstraction is a tool to bridge the divergence of human and machine.

The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.
– E. W. Dijkstra

```haskell
foo :: [Int] -> [Int]
foo = _
```

vs.

```haskell
foo :: Functor f => f Int -> f Int
foo = _
```

vs.

```haskell
foo :: a -> a
foo = id
```

But abstraction effectively *constrains* the number of primitives we can use

- Less primitives reduces the number of possible implementations
- Less possible implementations $=>$ less incorrect implementations

```haskell
cata :: Tree Int -> Int
cata = _
```

vs.

```haskell
cata :: Foldable f => f Int -> Int
cata = _
```

Concrete things do **NOT** compose!

```haskell
whatDoesItDo :: (
  ConnectionDB f,
  MonadReader Config f
) => IO ()
```

vs.

```haskell
whatDoesItDo :: Any -> ()
```

Maximum power
Minimal amount of reasoning

Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better.
– E. W. Dijkstra