Let the Types Work for You

Klarna Konferense

Felix Mulder

August 2018

What is this talk about?

What is its motivation?

And when will snacks be served?

Agenda

And when will snacks be served?

Agenda

- Functional Programming

And when will snacks be served?

Agenda

- Functional Programming
- Type systems

And when will snacks be served?

Agenda

- Functional Programming
- Type systems
- ???

And when will snacks be served?

Agenda

- Functional Programming
- Type systems
- ???
- Profit!

And when will snacks be served?

Agenda

- Functional Programming
- Type systems
- ???
- Profit!

Bio

And when will snacks be served?

Agenda

- Functional Programming
- Type systems
- ???
- Profit!

Bio

- Felix Mulder

And when will snacks be served?

Agenda

- Functional Programming
- Type systems
- ???
- Profit!

Bio

- Felix Mulder
- Software Engineer, IronBank

And when will snacks be served?

Agenda

- Functional Programming
- Type systems
- ???
- Profit!

Bio

- Felix Mulder
- Software Engineer, IronBank
- Compiler Engineer, Scala 3 @ EPFL

“Do you know that feeling of having to hold too many things in your head at once?”

Functional Programming gets rid of that by definition.

Game over, OO. Right?

What about the ???

The benefits are obvious

Referential Transparency + Types

==

Refactor All The Things! (without fear)

What about the downsides?

What if you could negate those downsides?

What if the compiler could write your program for you?

Today we’re exploring type-level induction and recursion

Coding time!

Constraints Liberate, and Liberties Constrain

`Any ⇒ Unit`

Felix's Conjecture

`Any ⇒ Unit`

Felix's Conjecture

> *"By being able to do anything, we can assume nothing"*

`Any ⇒ Unit`

Felix's Conjecture

> *"By being able to do anything, we can assume nothing"*

Constraints Liberate, and Liberties Constrain

```
def foo(i: Int): Int = ???
```

`Any ⇒ Unit`

Felix's Conjecture

> *"By being able to do anything, we can assume nothing"*

Constraints Liberate, and Liberties Constrain

```
def foo(i: Int): Int = ???
```

Constraints Liberate, and Liberties Constrain

```
def foo[A](a: A): A= ???
```

Any ⇒ Unit

Felix's Conjecture

> *"By being able to do anything, we can assume nothing"*

Constraints Liberate, and Liberties Constrain

```
def foo(i: Int): Int = ???
```

Constraints Liberate, and Liberties Constrain

```
def foo[A](a: A): A= ???
```

Constraints Liberate, and Liberties Constrain

```
def foo[A](a: A): A = a
```

Any ⇒ Unit

Felix's Conjecture

> *"By being able to do anything, we can assume nothing"*

Constraints Liberate, and Liberties Constrain

```
def foo(i: Int): Int = ???
```

Constraints Liberate, and Liberties Constrain

```
def foo[A](a: A): A= ???
```

Constraints Liberate, and Liberties Constrain

```
def foo[A](a: A): A = a
```

Constraints Liberate, and Liberties Constrain

“The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise”

– Edsger W. Dijkstra

In Closing

- Type level recursion for fun and profit!

In Closing

- Type level recursion for fun and profit!
- FP combined with sophisticated types only require edge validation

In Closing

- Type level recursion for fun and profit!
- FP combined with sophisticated types only require edge validation
- You don’t have to work against the compiler, make it work for you!