Refinement Types for Elm

By Lucas Payr

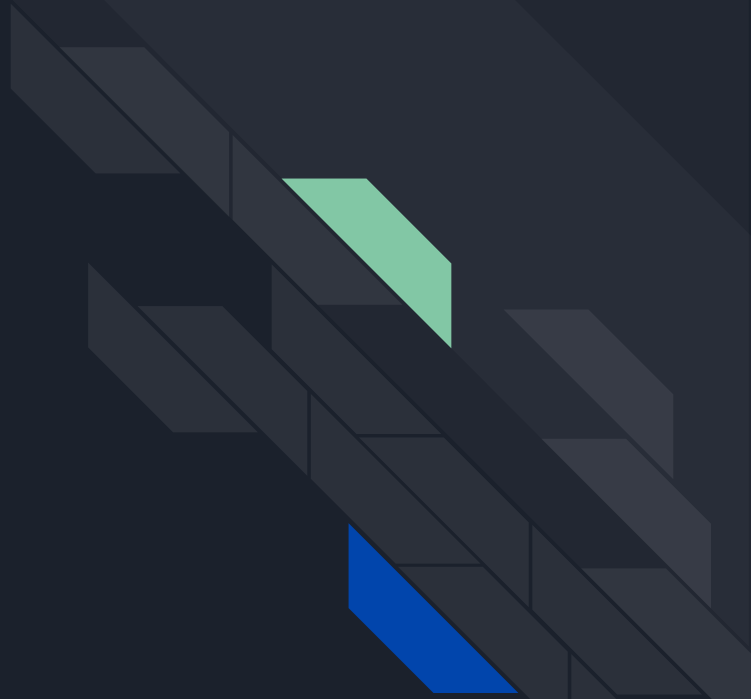
About me

- **Computer Mathemtics** student at the Johannes Kepler University in Linz.
- Currently working on my Master thesis: "**Refinement Types for Elm**"
  - Started working on it in Summer 2019
  - Expected completion in Summer 2021
- Master thesis has 3 parts:
  - Proving that every program written in Elm can be also compiled.
  - Proving that add Refinement types does not break anything.
  - Implementing a type checker for refinement types (As a fork of Elm-Analyse)

@orasund on Github

Refinement Types are

“verifications done in compile time”



Division by Zero

```
(//) : Int -> Int -> Int

1 // 0 --> 0   this is not good!
```

Wouldn't it be great if this could be caught in compile time?

```
(//) : Int -> NonZeroInt -> Int

1 // 0 --> COMPILER ERROR
```

Off By One Errors

```
type alias Model =
  { options : Array String
  , selected : Int --We actually mean an index
  }
```

Making impossible states impossible, right?

```
type alias Model =
  { options : Array String
  , selected : IntFromZeroToTen
  }
```

Pre- and Post-Conditions

```
max : Int -> Int -> Int
```

We might want to have a more specific type

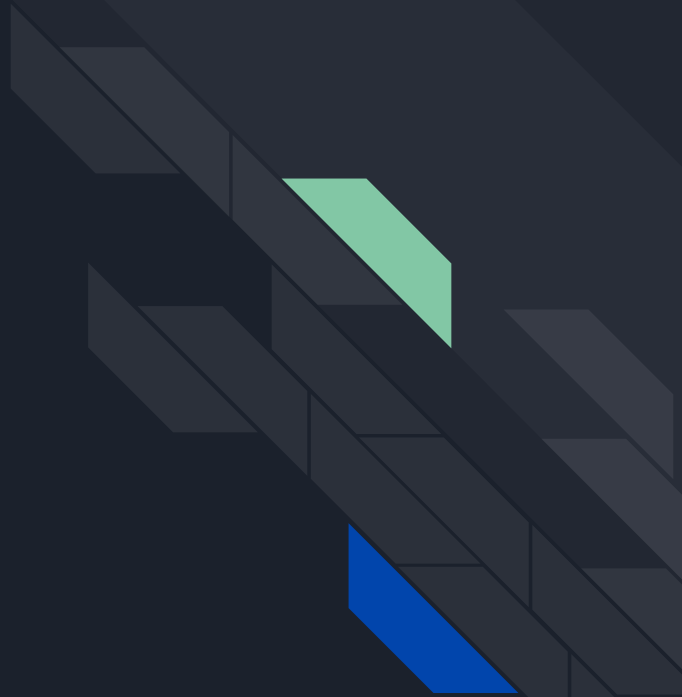
```
max : GivenTwoIntsReturnTheBiggerOne
```

How to Refinement types work?

Refinement types =

Type

\+ Refining Function



Using Refinement Types

```
{-| @refined \i -> i /= 0
-}
type alias NonZeroInt =
  Int
```

```
{-| @refined \i -> 10 >= i && i >= 0
-}
type alias IntFromZeroToTen =
  Int
```

```
{-| @refined
      \a b out -> (a == out && a >= b) || (b == out && b >= a)
-}
type alias GivenTwoIntsReturnTheBiggerOne =
  Int -> Int -> Int
```

Evan, why don’t you *just* add this to Elm?

😅...I might not have been completely honest...

The Downsides

- The compiler sometimes gets stuck
  - Even the fastest supercomputer can't compile every program that uses refinement types

So what do we do?

- We allow refinement types only on a subset where the compiler never gets stuck
- No multiplication of two variables!
- Use only **Int**, **Tuple of Ints** and **Function from Int to Ints**

Example

```
fun : IntFromZeroToTen -> IntFromZeroToTen
fun x =
  if (x * x) <= (x + x) then --compiler gets stuck
    x * x
  else
    x
```

```
fun : IntFromZeroToTen -> IntFromZeroToTen
fun x =
  if (x >= 0) && (x <= 2) then --compiler can continue
    x * x
  else
    x
```

Further Readings

- Liquid Haskell - Refinement types for Haskell
- Refined Scala - Refinement types for Scala

Thanks for your attention