Write You Some Proofs for Great Good

https://github.com/rpeszek/present-proofs-lc19

https://github.com/rpeszek/present-proofs-lc19/blob/master/doc/slides.pdf

Robert Peszek
r_peszek@hollandhart.co
m
Holland & Hart LLC
Innovation Lab

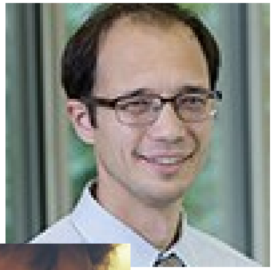
FP @ Innovation Lab, Holland & Hart

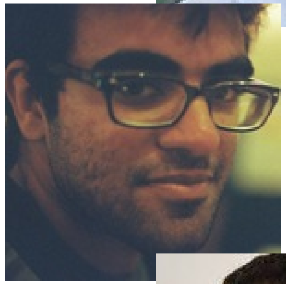
https://www.meetup.com/Boulder-Haskell-Programmers/

Jason Adaska - JWAdaska@hollandhart.com

Nihil Shah - NAShah@hollandhart.com

Robert Peszek - RPeszek@hollandhart.com





Why proofs?

- Curry Howard (Why proofs == Why programs)*
- Formal verification
- Enable `Type Precision` (*focus of this talk*)

```
data List a =
            Empty
            | Cons a  (List a)
```

vs

```
data Vect (n :: Nat) a  where
    Empty ::  Vect 'Z a
    Cons :: a -> Vect n a -> Vect ('S n) a
```

Plan for this talk

- Narrow scoped, pragmatic intro to proofs with dependent types
- Focus on what can be done vs how
- Anything wrong with List?
- Anything wrong with Maybe?
- Anything wrong with Bool?
- Q: Proofs and software engineering - together?
- Concerns
  - &#;Performance; Termination / Totality; Maintenance; …

Proofs by who writes them

	Programmers *	Automated Logic Solvers
Work		Free Lunch!
What can be done	A Lot	Limitted
Examples	Dependently Typed Languages	Refinement Types LiquidHaskell
	(focus of this talk)	

Motivation *(call-side safety)*

```
(!!) :: [a] -> Int -> a
```

```
safeGet :: [a] -> Int -> Maybe a

-- Liquid:
{-@ (!!) :: x: [a] -> {i:Nat | i < len x} -> a @-}
(!!) :: [a] -> Int -> a

-- Dep Typed:
(!!) ::  Vect n a -> Fin n -> a
(!!!) :: Vect n a -> SNat m -> MaybeB (m < n) a

(!!!!) : (n:Nat) -> (xs:List a) -> {auto ok: InBounds n xs} -> a
```

- Questionable improvement
- Refinements
- Precise types *(talk focus)*

… and much more

https://github.com/rpeszek/present-proofs-lc19/tree/master/src/Motivation

Motivation *(implementation safety)*

(code)

https://github.com/rpeszek/present-proofs-lc19/blob/master/src/Motivation/Exfer.idr

Intro

(code) Curry-Howard vs Imperative

https://github.com/rpeszek/present-proofs-lc19/blob/master/src/Present/AnIntro.hs

https://github.com/rpeszek/present-proofs-lc19/blob/master/src/Present/MaybeB.hs

Problem in Paradise - Questions

Should type checker know basic algebra?

`a || True <==> True  (Bool algebra)`

`a + b == b + a     (Nat, Int, Float ... algebra) ...`

Answers:

- Dependently Types Langs: **No** (*Programmers supply proofs*)

- Refinement Types / LiquidHaskell: **Yes** (*SMT solver does the work*)

Type Equality

```
data a :~: b where
    Refl  ::  a :~: a
```

typecheck Refl only if same types

```
test1    = Refl :: 5 :~: 5                  -- GOOD
test10  = Refl :: 4 :~: 5                   -- ERR


test2    = Refl :: 2 + 3 :~: 3 + 2     -- GOOD
test20 :: SNat n1 -> SNat n2 -> n1 + n2 :~: n2 + n1
test20 _ _ = Refl                            -- ERR
```

[(base) Data.Type.Equality]

Example Combinators

- library *over* op semantics
- "pattern-matching on a variable
  of type (a :~: b) produces a proof that a ~ b" - *haddoc*

```
sym :: (a :~: b) -> (b :~: a)

trans :: (a :~: b) -> (b :~: c) -> (a :~: c)

apply :: (f :~: g) -> (a :~: b) -> (f a :~: g b)

inner :: (f a :~: g b) -> (a :~: b)

castWith :: (a :~: b) -> a -> b

gcastWith :: (a :~: b) -> ((a ~ b) => r) -> r
gcastWith Refl x = x
```

math *useful and tedious*

a bit unexpected ?

fullfil ~ constraint using :~:

[(base) Data.Type.Equality]()

Proofs - Bool Algebra

(code)

https://github.com/rpeszek/present-proofs-lc19/blob/master/src/Present/ProofsBoolAlg.hs

Better Bool ... Decidability

```
import Data.Void


data Dec prop = Yes prop |
                No (prop -> Void)
```

https://github.com/rpeszek/present-proofs-lc19/blob/master/src/Present/ProofsDecidable.hs

Proofs - Nat (Performance)

```
reverse :: Vect n a -> Vect n a
reverse = ...


-- recursive calls
plusCommutative :: SNat left -> SNat right -> ((left + right) :~: (right + left))
plusCommutative left right = case left of
          SZ  -> lemma1 right
          (SS k)  -> case plusCommutative k right of Refl -> sym (lemma2 right k)
```

unsafeCoerce replacements

https://github.com/rpeszek/present-proofs-lc19/blob/master/src/Present/ProofsNatAlg.hs

Some Learning/References

- intro books
  - Type Driven Development in Idris great book
    - https://github.com/rpeszek/IdrisTddNotes/wiki
  - TAPL great book *(not really dep types but still)*
  - Programming foundation books (penn/Pierce et al, Wadler)
- Haskell projects (with reading references)
  - singletons
  - equational-reasoning-in-haskell
  - liquidhaskell
- blogs
  - blog.jle.im (Justin Le)
  - typesandkinds (Richard Eisenberg)
- youtube
  - Introduction to Agda series by Daniel Peebles published by Edward Kmett