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.com
Holland & Hart LLC
Innovation Lab

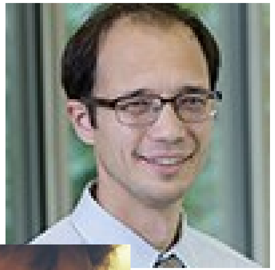
FP @ Innovation Lab, Holland & Hart

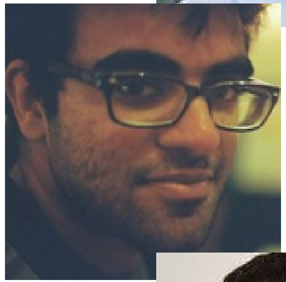
https://www.meetup.com/Boulder-Haskell-Programmers/

Jason Adaska - JWAdaska@hollandhart.com

Nihil Shah - NAShah@hollandhart.com

Robert Peszek - RPeszek@hollandhart.com





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)	

Plan for this talk

Narrow scoped, pragmatic intro to proofs with dependent types

- Mathematics and Software Engineering
  - Type Precision - Gentlemen's agreements
  - Progs == Proofs - Software == has Bugs
  - Dependent Types - Implementation Hiding
  - Proofs - Maintenance cost
- Type Precision: better List, better Maybe, better Bool, …
- Termination / Totality
- Performance
- Peano Nat vs GHC.TypeLits

Why proofs?

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

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

vs

```
data Vect (n :: Nat) a  where
     Empty ::  Vect 0 a
     Cons :: a -> Vect n a -> Vect (1 + n) a
```

```
id_sanity  ::  Vect (1 + n) a -> Vect (n + 1) a
id_sanity = unsafeCoerce                          -- ghrrr ☹
```

Curry-Howard to MaybeB

(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

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

(code)

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

- Programming contradictions
  - -XEmptyCase
- Forced Type Precision
- Programs are proofs! *

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

Proofs - Nat (Performance)

(code)

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

-- 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

- rewrite rules
- proveFast combinators
- CPP

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

Proofs - working with TypeLits

(code)

- hidden implementation
- Peano approach is slow

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

Maintenance

```
id_sanity  ::  SNat n -> Vect (1 + n) a -> Vect (n + 1) a
id_sanity n = case plusCommutative 1 n of Refl -> id
```

- (+) is just a function lifted to type level
- (+) implementation details get lifted too
- plusCommutative proof has to depend on (+) implementation
- change implementation of (+) and
  - Types change (compilation breaks)
  - Proofs break
- Gained Type Safety: Priceless

Type Precision *(call-side safety)*

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

- a lie ...

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

- questionable improvement
- refinements

... still imprecise *

```
isElem : DecEq a => (x: a) -> (xs: List a) -> Dec (Elem x xs)
```

not in the same boat *

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

Type Precision *(implementation safety)*

(code)

https://github.com/rpeszek/present-proofs-lc19/tree/master/src/Wrapup/Exfer.idr

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