An Introduction to Dependently Typed Functional Programming in Idris

Mark Farrell

April 21, 2015

Who am I?

What is Idris?

A Problem

```
data List : Type -> Type where
  Nil : List a
  (::) : a -> List a -> List a
```

```
head : List a -> a
head (x :: xs) = x
```

```
> head Nil
Error!
```

A Solution with Dependent Types

```
data Vect : Nat -> Type -> Type where
  Nil : Vect Z a
  (::) : a -> Vect n a -> Vect (S n) a
```

```
data Nat = Z | S Nat
```

```
head : Vect (S n) a -> a
head (x :: xs) = x
```

```
> head Nil
Safe: won't type check.
```

Install Idris

Another Problem

```
(++) : Vect n a ->
       Vect m a ->
       Vect (n + m) a
```

```
A silly implementation that type checks.
```

```
> [1] ++ [2, 3]
[2, 3, 1] : Vect 3 Nat
```

```
Oops!
```

Another solution with Dependent Types

```
v : Vect n a -> (v ++ Nil) = v
```

```
v : Vect n a -> (Nil ++ v) = v
```

```
v : Vect n a ->
w : Vect n a ->
x : Vect n a ->
(v ++ w) ++ x = v ++ w ++ x
```

```
v : Vect n a ->
w : Vect n a ->
x : Vect n a ->
v ++ (w ++ x) = v ++ w ++ x
```

```
Proofs of these propositions.
```

Simple Proof Examples

```
fiveIsFive : 5 = 5
fiveIsFive = Refl
```

```
-- lemma
cong : (a = b) -> f a = f b
```

```
-- lemma
plusZeroRightNeutral :
  (left : Nat) -> left + Z = left
```

```
twicedNeutral :
  (n : Nat) -> mult 2 n = plus n n
twicedNeutral n =
  cong (plusZeroRightNeutral n)
```

Exercises

```
tail : Vect (S n) a -> Vect n a
```

```
push : a -> Vect n a -> Vect (S n) a
```

```
rotations : Vect n a -> Vect n (Vect n a)
```

```
insertions : a -> Vect n a
  -> Vect (S n) (Vect (S n) a)
```

```
-- challenge
permutations : Vect n a
  -> Vect (fact n) (Vect n a)
```

```
(x : Nat) -> (y : Nat) -> (z : Nat)
  -> (x + y) + z = x + y + z
```

```
(x : Nat) -> (y : Nat) -> (z : Nat)
  -> x + (y + z) = x + y + z
```

```
-- Tip: consult the proof wiki.
(x : Nat) -> (y : Nat) -> x + y = y + x
```

Further Reading

Concluding Remarks

Questions?