A Dependently-Typed Zipper over GADT-Embedded ASTs

Donovan Crichton

November 2018

Preliminaries

- **Slides and examples available at:** `https://github.com/donovancrichton/talkdepzip.git`
- **About me:**
  - Honours 'year' student at Griffith University.
  - Working towards a type-correct genetic program through dependently-typed functional programming.
  - About 18 months experience in FP, just under 12 with dependent types.

A refresher on dependent types 1.

- The most basic definition is a dependent data type (GADT in Haskell).
- Dependent data-types depend on being parameterised over a value for their construction.
- Distinguished from parameterised ADTs by the ability to specify the return type parameter of each data constructor.

A vector dependent on a length value.

```
data Nat = Z | S Nat

data Vec : (n : Nat) -> (e : Type) -> Type where
  Nil : Vec Z e
  (::) : (x : e) -> (xs : Vec n e) -> Vec (S Z) e
```

A refresher on dependent types 2.

Why is this good?

If our length forms part of our type, we gain the ability to write correct functions with respect to vector length, without having to explicitly check.

Adding some vectors.

```
-- total
(+) : Num a => Vect n a -> Vect n a -> Vect n a
(+) [] [] = []
(+) (x :: xs) (y :: ys) = x + y :: xs + ys
```

A refresher on dependent types 3.

Π types.

- The Π type is a family of types that are indexed by a value (hence type families in Haskell).
- Π types are used to calculate correct return types when given a specified value.
- In Idris Π types only fully refine if the functions requiring them are marked as total.
- In Idris functions that return Π types don't always refine in function composition, recursive calls or let bindings.

A refresher on dependent types 4.

An example of using Π types in Idris.

```
Age : Type
Age = Nat

Name : Type
Name = String

data Material = Plastic | Wood | Metal | Cheese
data Person = P Name Age
data Object = O Material

IsPerson : Bool -> Type
IsPerson True = Person
IsPerson False = Object

isPerson : (x : Bool) -> IsPerson x
isPerson True = P "Donovan Crichton" 33
isPerson False = O Cheese
```

A refresher on dependent types 5.

Σ types.

- Σ types are a pairing of a value, and a type that depends on that value (They are also called dependent pairs).
- Σ types are useful when you want some basic type inference around dependent types!
- In Idris it is difficult to extract either size of a Σ type pair. Particularly when that pair is under a constructor.
- Due to the dependent nature of the `fst` and `snd` functions on dependent pairs, they cannot be used with ordinary maps, folds, binds etc.

A refresher on dependent types 6.

An example using Σ types in Idris.

```
data DPair : (a : Type) -> (P : a -> Type) -> Type where
  MkDPair : (x : a) -> (pf : P x) -> DPair a P

-- also has some syntactic sugar in Idris.
f : (x : Bool ** IsPerson x)
f = (_ ** isPerson True)

g : Num a => (n : Nat ** Vec n a)
g = (_ ** [1, 2, 3])

-- this is particularly useful if we are passing a vector
-- of unknown length in as an argument.
len : Num a => Vec n a -> (x : Nat ** Vec x a)
len x = (_ ** x)

-- len [1, 2, 3] returns
-- (3 ** [1, 2, 3]) : (x : Nat ** Vec x Integer)
```

A refresher on dependent types 7.

Why are Σ and Π good?

- It turns out that the curry-howard isomorphsim still holds with the introduction of dependent types.
- Lets say we have some proof $P\ x$, using a dependent-data-type or a Π type is saying $\forall x, P\ x$. Using a Σ type is saying $\exists x, P\ x$.
- This lets us create more expressive types in order to restrict the number of invalid programs even further!

Dependent data-type embedded DSLs