Structured Traversals for (Multiply) Recursive Algebraic Datatypes

G. Cassian Alexandru

January 11, 2021

Presentation generated from `.lhs` sources using `lhs2TeX`

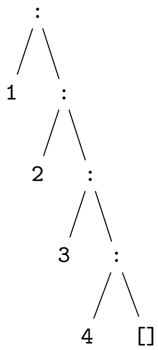
```haskell
length :: [a] → Int
length [] = 0
length (x:xs) = 1 + length xs

filter :: (a → Bool) → [a] → [a]
filter p = go where
  go [] = []
  go (x:xs) = if p x then [x] else [] ++ go xs
```

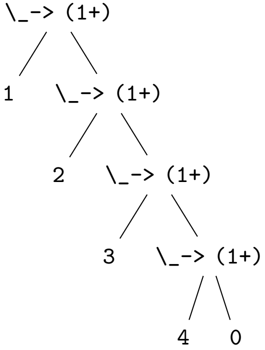
- List Design pattern?
- Design Patterns are a poor man's abstraction
- Recognize common structure & find correct abstract notion

Traversals

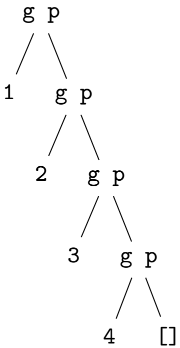
List



`length`



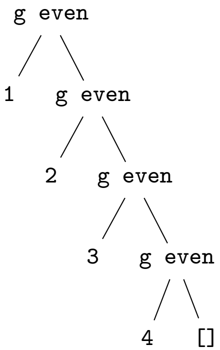
`filter p`




```
g p x xs =
  bool [] [x] (p x) ++ xs
```

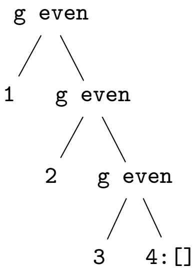
Example Evaluation of `filter even`

```
g p x xs =
  bool [] [x] (p x) ++ xs
```



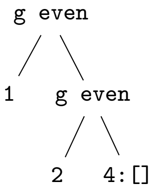
Example Evaluation of `filter even`

```
g p x xs =
  bool [] [x] (p x) ++ xs
```



Example Evaluation of `filter even`

```
g p x xs =
  bool [] [x] (p x) ++ xs
```



Example Evaluation of `filter even`

```
g p x xs =
  bool [] [x] (p x) ++ xs
```

```
  g even
   /  \
 1    2:4:[]
```

Example Evaluation of `filter even`

```
g p x xs =
  bool [] [x] (p x) ++ xs
```

```
2:4:[]
```

```
data List a = Nil | Cons a (List a)
data BooL = TT | FF
```

GADT Syntax:

```
data List a where
  Nil :: List a
  Cons :: a → (List a) → (List a)
data BooL where
  TT :: BooL
  FF :: BooL
```

```
data List a where
  Nil :: List a
  Cons :: a → (List a) → (List a)
data Bool where
  TT :: Bool
  FF :: Bool

list :: b → (a → b → b) → List a → b
list nil cons = fold where
  fold Nil = nil
  fold (x `Cons` xs) = x `cons` fold xs

bool’ :: b → b → Bool → b
bool’ tt ff = fold where
  fold TT = tt
  fold FF = ff
```

Algebras

$F : \mathcal{C} \to \mathcal{C}, A, B, A_0 \in \mathcal{C}_0$

Algebra

$$\begin{tikzcd} FA \arrow[d, "\phi"] \\ A \end{tikzcd}$$

Algebra-Hom:
$(A, \phi) \to (B, \psi)$

$$\begin{tikzcd} FA \arrow[r, "Ff"] \arrow[d, "\phi"] & FB \arrow[d, "\psi"] \\ A \arrow[r, "f"] & B \end{tikzcd}$$

Initial Algebra:
(A, κ)

$$\begin{tikzcd} FA \arrow[r, dashed, "Fh"] \arrow[d, shift left, "\kappa"] & FB \arrow[d, "\psi"] \\ A \arrow[u, shift left, "\kappa^{-1}"] \arrow[r, dashed, "h"] & B \end{tikzcd}$$

Initiality requirement: $h = \kappa^{-1}; Fh; \psi$

As Program

```haskell
newtype Fix f = In { out :: f (Fix f) }

type Algebra f c = f c → c

cata :: Functor f => Algebra f a → Fix f → a
cata alg = alg · fmap (cata alg) · out
```