

# Recursive types for free! in Haskell<sup>1</sup>

Traian Florin Șerbănuță

University of Bucharest and Runtime Verification

---

<sup>1</sup>Based on Philip Wadler's *Recursive types for free!*

## GHC language extensions used

```
{-# LANGUAGE DeriveFunctor           #-}  
{-# LANGUAGE ExistentialQuantification #-}  
{-# LANGUAGE ExplicitForAll         #-}  
{-# LANGUAGE GADTs                  #-}  
{-# LANGUAGE Rank2Types              #-}
```

## Definitions of f-algebras and f-coalgebras

An F-algebra is a pair  $(X, k)$  consisting of an object  $X$  and an arrow  $k : F X \rightarrow X$ .

`type Algebra f x = f x -> x`

A morphism between  $(X, k)$  and  $(X', k')$  is given by an arrow  $h : X \rightarrow X'$  such that the following diagram commutes.

$$(1) \quad \begin{array}{ccc} & & k \\ & F X & \xrightarrow{\quad\quad\quad} X \\ & | & | \\ & | & | \\ F h & | & | h \\ & | & | \\ & v & v \\ & F X' & \xrightarrow{\quad\quad\quad} X' \\ & & k' \end{array}$$

These form a category.

## Definition of f-coalgebras

An F-coalgebra is a pair  $(X, k)$  consisting of an object  $X$  and an arrow  $k : X \rightarrow F X$ .

```
type CoAlgebra f x = x -> f x
```

A morphism between  $(X, k)$  and  $(X', k')$  is given by an arrow  $h : X \rightarrow X'$  such that the following diagram commutes.

$$\begin{array}{ccc} & k & \\ & \text{-----} & \\ X & \longrightarrow & F X \\ | & & | \\ | & & | \\ h | & & | F h \\ | & & | \\ v & & v \\ X' & \xrightarrow{k'} & F X' \end{array}$$

These form a category.

## Least-fixpoints as (weak) initial algebras

```
newtype LFix f =  
  LFix { unLFix :: (forall x . Algebra f x -> x) }
```

- ▶ `LFix f` embodies the idea of a type for terms associated to `f`.
- ▶ A term can be (uniquely) evaluated in any algebra.
- ▶ A term gives, for an algebra, a value for the term in the algebra.
- ▶ Whence the type for a term: `forall x . Algebra f x -> x`

## Least-fixpoints as (weak) initial algebras

```
newtype LFix f =  
  LFix { unLFix :: (forall x . Algebra f x -> x) }
```

		wInitialAlg	
	f (LFix f)	----->	LFix f
fmap (fold algebra)			fold algebra
	v		v
	f a	----->	a
			algebra

```
fold :: Algebra f a -> LFix f -> a
```

```
fold algebra term = unLFix term algebra
```

```
weakInitialAlgebra :: Functor f => Algebra f (LFix f)
```

```
weakInitialAlgebra s =
```

```
  LFix ( \alg -> alg (fmap (fold alg) s) )
```

## Morphism condition for fold algebra

$$\begin{array}{ccc} & \text{wInitialAlg} & \\ f \text{ (LFix } f) & \xrightarrow{\quad\quad\quad} & \text{LFix } f \\ & | & | \\ \text{fmap (fold algebra)} & | & | \text{ fold algebra} \\ & | & | \\ & v & v \\ & f \text{ a} \xrightarrow{\quad\quad\quad} & a \\ & \text{algebra} & \end{array}$$

```
(fold algebra . weakInitialAlgebra) fterm
== fold algebra (weakInitialAlgebra fterm)
== unLFix (LFix (\alg -> alg (fmap (fold alg) fterm)))
   algebra
== algebra (fmap (fold algebra) fterm)
== (algebra . fmap (fold algebra)) fterm
```

## Natural numbers as a least fix point

```
data NatF x = Zero | Succ x deriving Functor

type Nat = LFix NatF

zero :: Nat
zero = weakInitialAlgebra Zero

successor :: Nat -> Nat
successor n = weakInitialAlgebra (Succ n)

one :: Nat
one = successor zero

integral :: Integral n => Algebra NatF n
integral Zero      = 0
integral (Succ x) = x + 1

natToIntegral :: Integral n => Nat -> n
natToIntegral = fold integral
```



## Lists as a least fix point

```
data ListF a x = Nil | LCons a x deriving Functor

type List a = LFix (ListF a)

nil :: List a
nil = weakInitialAlgebra Nil

cons :: a -> List a -> List a
cons a l = weakInitialAlgebra (LCons a l)

list :: Algebra (ListF a) [a]
list Nil          = []
list (LCons a l) = a:l

toList :: List a -> [a]
toList = fold list
```

## When is the least fix point actually initial?

$$\begin{array}{ccc} & \text{alg} & \\ f \ X & \xrightarrow{\quad\quad\quad} & X \\ | & & | \\ | & & | \\ f \ h & | & | \ h \quad \text{implies} \quad \text{id} \ | \\ | & | & | \\ v & v & v \\ f \ X' & \xrightarrow{\quad\quad\quad} & X' \\ & \text{alg}' & \end{array} \qquad \begin{array}{ccc} & \text{fold alg} & \\ \text{LFix } f & \xrightarrow{\quad\quad\quad} & X \\ | & & | \\ | & & | \\ \text{id} \ | & & | \ h \\ | & & | \\ v & & v \\ \text{LFix } f & \xrightarrow{\quad\quad\quad} & X' \\ & \text{fold alg}' & \end{array}$$

$h: (X, \text{alg}) \rightarrow (X', \text{alg}')$  implies  $h \cdot \text{fold alg} == \text{fold alg}'$

Additionally,  $\text{fold weakInitialAlgebra} == \text{id}$

## Initiality consequences

If  $(\text{LFix } f, \text{weakInitialAlgebra})$  is initial, then  $\text{weakInitialAlgebra}$  is an isomorphism and its inverse is:

```
weakInitialAlgebraInv :: Functor f => CoAlgebra f (LFix f)
weakInitialAlgebraInv = fold (fmap weakInitialAlgebra)
```

$$\begin{array}{ccc} f \text{ (LFix } f) & \xrightarrow{\quad\quad\quad} & \text{LFix } f \\ | & \text{wInitialAlg} & | \\ | & & | \\ f \text{ wInitialAlgInv} & | & | \text{ wInitialAlgInv} \\ | & & | \\ v & f \text{ wInitialAlg} & v \\ f \text{ (f (LFix } f)) & \xrightarrow{\quad\quad\quad} & f \text{ (LFix } f) \\ | & & | \\ | & & | \\ f \text{ wInitialAlg} & | & | \text{ wInitialAlg} \\ | & & | \\ v & \text{wInitialAlg} & v \\ f \text{ (LFix } f) & \xrightarrow{\quad\quad\quad} & \text{LFix } f \end{array}$$

## Greatest fix points as (weak) final co-algebras

```
data GFix f = forall x . GFix (CoAlgebra f x, x)
```

		coalg	
	X	----->	f X
unfold coalg			f (unfold coalg)
	GFix f	----->	f (GFix f)
		wFinalCoalg	

```
unfold :: CoAlgebra f a -> a -> GFix f
```

```
unfold coalg a = GFix (coalg, a)
```

```
weakFinalCoAlgebra :: Functor f => CoAlgebra f (GFix f)
```

```
weakFinalCoAlgebra (GFix (coalg, a)) = fmap (unfold coalg)
```

## When is the greatest fix point actually final?

$$\begin{array}{ccc} & \text{alg} & \\ X & \xrightarrow{\quad\quad\quad} f\ X & \\ | & & | \\ | & & | \\ h\ | & & | \quad F\ h \quad \text{implies} \quad h\ | \\ | & & | \\ v & & v \\ X' & \xrightarrow{\quad\quad\quad} f\ X' & \\ & \text{alg}' & \end{array} \qquad \begin{array}{ccc} & \text{unfold alg} & \\ X & \xrightarrow{\quad\quad\quad} \text{GFix } f & \\ | & & | \\ | & & | \\ h\ | & & | \quad \text{id} \\ | & & | \\ v & & v \\ X' & \xrightarrow{\quad\quad\quad} \text{GFix } f & \\ & \text{unfold alg}' & \end{array}$$

```
weakFinalCoAlgebraInv :: Functor f => Algebra f (GFix f)
weakFinalCoAlgebraInv = unfold (fmap weakFinalCoAlgebra)
```

## Streams as a greatest fix point

```
data StreamF a x = SCons { headF :: a, tailF :: x }  
    deriving Functor  
type Stream a = GFix (StreamF a)  
  
headS :: Stream a -> a  
headS = headF . weakFinalCoAlgebra  
  
tailS :: Stream a -> Stream a  
tailS = tailF . weakFinalCoAlgebra  
  
stream :: CoAlgebra (StreamF a) [a]  
stream (a:as) = SCons a as  
  
toStream :: [a] -> Stream a  
toStream = unfold stream  
  
type IStream a = LFix (StreamF a)  
  
icons :: a -> IStream a -> IStream a  
icons a s = weakInitialAlgebra (SCons a s)
```

## Introducing the recursion schemes Fix construction

```
newtype Fix f where
  Fix :: f (Fix f) -> Fix f   -- Fix is an f-algebra

unFix :: Fix f -> f (Fix f)   -- unFix is an f-coalgebra
unFix (Fix x) = x

cata :: Functor f => Algebra f a -> Fix f -> a
cata alg = go
  where
    go = alg . fmap go . unFix

ana :: Functor f => CoAlgebra f a -> a -> Fix f
ana coalg = go
  where
    go = Fix . fmap go . coalg
```

## Relating Fix with LFix and GFix

```
lFixToFix :: LFix f -> Fix f
```

```
lFixToFix = fold Fix
```

```
fixToLFix :: Functor f => Fix f -> LFix f
```

```
fixToLFix = cata weakInitialAlgebra
```

```
fold Fix . cata weakInitialAlgebra
```

```
= Fix . fmap (fold Fix . cata weakInitialAlgebra) . unFix
```

```
fixToGFix :: Functor f => Fix f -> GFix f
```

```
fixToGFix = unfold unFix
```

```
gFixToFix :: Functor f => GFix f -> Fix f
```

```
gFixToFix = ana weakFinalCoAlgebra
```



## Read More

- ▶ Philip Wadler (1990) Recursive types for free!
- ▶ Bartosz Milewski (2013) Understanding F-Algebras
- ▶ A formalization of the above in Coq with actual proofs