Least fixed point (Data)

```
(* → *) → *
μf = f (μf) = f (… (f 0)) = Free f 0
```

Instances of μf are “f-data-structures” or short “f-structures”.

Free Monads

```
(* → *) → * → *
Free f a = a + f (Free f a)
           variable   term
```

A finite f-structure, that can contain as. Is a functor and a monad. Monadic-bind corresponds to substitution: Substitutes as by terms that can contain bs.

Greatest fixed point (Codata)

```
(* → *) → *
νf = f (νf) = f (f (…)) = Cofree f 1
```

Instances of νf are “f-codata-structures” or short “f-structures”.

Cofree Comonads

```
(* → *) → * → *
Cofree f a = a , f (Free f a)
        annotation   trace
```

A possibly infinite f-structure, full of as. Is a functor and a comonad. Comonadic-extend corresponds to computing a new f-structure full of bs. At every level the a and the full trace are available for computing the b.

Destruction Morphisms

catamorphism

```
cata :: ∀ a. (f a → a) → μf → a
              f-algebra
```

Also known as “fold”. Deconstructs a f-structure level-by-level and applies the algebra [13, 5, 14, 6].

paramorphism

```
para :: ∀ a. (f (μf , a) → a) → μf → a
```

A.k.a. “the Tupling-Trick”. Like cata, but allows access to the full subtree during teardown. Is a special case of zygo, with the helper being the initial-algebra [16].

zygomorphism

```
zygo :: ∀ a b. (f (a , b) → a) →
               (f b → b)       → μf → a
```

Allows depending on a helper algebra for deconstructing a f-structure. A generalisation of para.

histomorphism

```
histo :: ∀ a. (f (Cofree f a) → a) → μf → a
```

Deconstructs the f-structure with the help of all previous computation for the substructures (the trace). Difference to para: The subcomputation is already available and needs not to be recomputed.

prepromorphism

```
prepro :: ∀ a. (f a → a) → (f ⇝ f) → μf → a
```

Applies the natural transformation at every level, before destructing with the algebra. Can be seen as a one-level rewrite. This extension can be combined with other destruction morphisms [4].

Construction Morphisms

anamorphism

```
ana :: ∀ a. (a → f a) → a → νf
             f-coalgebra
```

Also known as “unfold”. Constructs a f-structure level-by-level, starting with a seed and repeatedly applying the coalgebra [13, 5].

apomorphism

```
apo :: ∀ a.  (a → f (a + νf)) → a → νf
```

A.k.a. “the Co-Tupling-Trick”™. Like ana, but also allows to return an entire substructure instead of one level only. Is a special case of g-apo, with the helper being the final-coalgebra [17, 16].

g-apomorphism

```
gapo :: ∀ a b. (a → f (a + b)) →
               (b → f b)       → a → νf
```

Allows depending on a helper coalgebra for constructing a f-structure. A generalisation of apo.

futumorphism

```
futu :: ∀ a. (a → f (Free f a)) → a → νf
```

Constructs a f-structure stepwise, but the coalgebra can return multiple layers of a-valued substructures at once. Difference to apo: the subtrees can again contain as [16].

postpromorphism

```
postpro :: ∀ a. (a → f a) → (f ⇝ f) → a → νf
```

Applies the natural transformation at every level, after construction with the coalgebra. Can be seen as a one-level rewrite. This extension can be combined with other construction morphisms.

Combined Morphisms

ana then cata = hylomorphism

```
hylo :: ∀ a b. (a → f a) → (f b → b) → a → b
```

Omits creating the intermediate structure and immediately applies the algebra to the results of the coalgebra† [13, 2, 5, 14].

ana then histo = dynamorphism

```
dyna :: ∀ a b. (a → f a) →
               (f (Cofree f b) → b) → a → b
```

Constructs a structure and immediately destructs it while keeping intermediate results†. Can be used to implement dynamic-programming algorithms [9, 10].

futu then histo = chronomorphism

```
chrono :: ∀ a b. (a → (Free f a)) →
                 (f (Cofree f b) → b) → a → b
```

Can at the same time “look back” at previous results and “jump into the future” by returning seeds that are multiple levels deep† [11].

cata then conv then ana = metamorphism

```
meta :: ∀ a b. (f a → a) → (a → b) → (b → g b) →
               μf → νg
```

Constructs a g-structure from a f-structure while changing the internal representation in-between [7].

Other Morphisms

Most of the above morphisms can be modified to accept generalized algebras (with w being a comonad)

```
GAlgebra f w a = f (w a) → a
```

or generalised coalgebras (with m being a monad), respectively:

```
GCoalgebra f m a = a → f (m a)
```

Also a multitude of other morphisms exist [12, 3, 1] and the combination of morphisms and distributive laws

```
Distr f g = ∀ a. f (g a) → g (f a)
```

has been studied [8, 15].

† Can also be enhanced by a representation change (natural transformation f ⇝ g), before deconstructing with a corresponding g-algebra

[1] Adámek, Jiří, Stefan Milius, and Jiří Velebil. "Elgot algebras." *Electronic Notes in Theoretical Computer Science,* 2006.

[2] Augusteijn, Lex. "Sorting morphisms." *Advanced Functional Programming. Springer Berlin Heidelberg*, 1998.

[3] Erwig, Martin. Random access to abstract data types. *Springer Berlin Heidelberg*, 2000.

[4] Fokkinga, Maarten M. "Law and order in algorithmics.” *PhD Thesis,* 1992.

[5] Gibbons, Jeremy. "Origami programming.”, 2003.

[6] Gibbons, Jeremy. "Design patterns as higher-order datatype-generic programs.” *Proceedings of the Workshop on Generic programming*. ACM, 2006.

[7] Gibbons, Jeremy. "Metamorphisms: Streaming representation-changers." *Science of Computer Programming*, 2007.

[8] Hinze, Ralf, et al. "Sorting with bialgebras and distributive laws." *Proceedings of the Workshop on Generic programming*. ACM, 2012.

[9] Hinze, Ralf, and Nicolas Wu. "Histo-and dynamorphisms revisited." *Proceedings of the Workshop on Generic programming*. ACM, 2013.

[10] Kabanov, Jevgeni, and Varmo Vene. "Recursion schemes for dynamic programming.” *Mathematics of Program Construction*. Springer Berlin Heidelberg, 2006.

[11] Kmett, Edward. “Time for Chronomorphisms.”, 2008. http://comonad.com/reader/2008/time-for-chronomorphisms/

[12] Kmett, Edward. “Recursion Schemes: A Field Guide (Redux).”, 2009. http://comonad.com/reader/2009/recursion-schemes/

[13] Meijer, Erik, Maarten Fokkinga, and Ross Paterson. "Functional programming with bananas, lenses, envelopes and barbed wire." *Functional Programming Languages and Computer Architecture*. Springer Berlin Heidelberg, 1991.

[14] Oliveira, Bruno, and Jeremy Gibbons. "Scala for generic programmers." *Proceedings of the Workshop on Generic programming*. ACM, 2008.

[15] Turi, Daniele, and Gordon Plotkin. "Towards a mathematical operational semantics." *Logic in Computer Science.* IEEE, 1997.

[16] Uustalu, Tarmo, and Varmo Vene. "Primitive (co) recursion and course-of-value (co) iteration, categorically." *Informatica*, 1999.

[17] Vene, Varmo, and Tarmo Uustalu. "Functional programming with apomorphisms (corecursion)." *Proceedings of the Estonian Academy of Sciences: Physics, Mathematics*. Vol. 47. No. 3. 1998.