

Chapter 29: Hylomorphism and Metamorphism

Zhenjiang Hu, Wei Zhang

School of Computer Science
Peking University

December 26, 2025

Hylomorphism: fold . unfold

A **hylomorphism** is a fold after an unfold.

$$\mathit{hylo} (\oplus, e) (p, f, g) = \mathit{foldr} (\oplus) e \cdot \mathit{unfold} p f g$$

Hylomorphism: fold . unfold

A **hylomorphism** is a fold after an unfold.

$$\mathit{hyla} (\oplus, e) (p, f, g) = \mathit{foldr} (\oplus) e \cdot \mathit{unfold} p f g$$

The unfold phase **produces** an intermediate **finite list** (we leave discussion on infinite lists and other types later), while the fold phase **consumes** it to produce the result.

Hylomorphism: fold . unfold

A **hylomorphism** is a fold after an unfold.

$$\mathit{hylo} (\oplus, e) (p, f, g) = \mathit{foldr} (\oplus) e \cdot \mathit{unfold} p f g$$

The unfold phase **produces** an intermediate **finite list** (we leave discussion on infinite lists and other types later), while the fold phase **consumes** it to produce the result.

We can define *hylo* recursively as follows (without generating the intermediate list).

$$\frac{\mathit{hylo} (\oplus, e) (p, f, g) x =}{\text{if } p x \text{ then } e \text{ else } f x \oplus \mathit{hylo} (\oplus, e) (p, f, g) (g x)}$$

Specification with *hylo*

Hylomorphism is useful to specify functions where lists are used during computation, where lists may be neither input nor output.

Specification with *hylo*

Hylomorphism is useful to specify functions where lists are used during computation, where lists may be neither input nor output.

Example: *factorial* in *hylo*

The factorial function

$$\textit{factorial } n = n!$$

Specification with *hylo*

Hylomorphism is useful to specify functions where lists are used during computation, where lists may be neither input nor output.

Example: *factorial* in *hylo*

The factorial function

$$\mathit{factorial} \ n = n!$$

can be easily specified as a composition of unfold and fold:

$$\mathit{factorial} \ n = \mathit{product} \cdot \mathit{upto} \ (1, n).$$

Specification with *hylo*

Hylomorphism is useful to specify functions where lists are used during computation, where lists may be neither input nor output.

Example: *factorial* in *hylo*

The factorial function

$$\mathit{factorial} \ n = n!$$

can be easily specified as a composition of unfold and fold:

$$\mathit{factorial} \ n = \mathit{product} \cdot \mathit{upto} \ (1, n).$$

That is,

$$\mathit{factorial} \ n = \mathit{hylo} \ (\times, 1) \ (\mathit{fstGreater}, \mathit{fst}, \mathit{fstSucc}) \ n.$$

Specification with *hylo*

Exercise: L4-1

Define as a *hylo* the function $sumSquares(m, n)$ that sums up the squares of numbers from m to n .

$$sumSquares(2, 5) = 2^2 + 3^2 + 4^2 + 5^2 = 54$$

Specification with *hylo*

Exercise: L4-1

Define as a *hylo* the function *sumSquares* (m, n) that sums up the squares of numbers from m to n .

$$\text{sumSquares}(2, 5) = 2^2 + 3^2 + 4^2 + 5^2 = 54$$

Exercise: L4-2

Define as a *hylo* the function to compute all even Fibonacci numbers that are less than 100,000.

Metamorphism: unfold . fold

While a hylomorphism is a fold after an unfold, a **metamorphism** is the opposite composition; an unfold after a fold.

$$\mathit{meta} (p, f, g) (\oplus, e) = \mathit{unfold} p f g \cdot \mathit{foldr} (\oplus) e$$

Metamorphism: unfold . fold

While a hylomorphism is a fold after an unfold, a **metamorphism** is the opposite composition; an unfold after a fold.

$$\mathit{meta} (p, f, g) (\oplus, e) = \mathit{unfold} p f g \cdot \mathit{foldr} (\oplus) e$$

A list is consumed by a fold, resulting in a value for producing a new list by an unfold.

Specification with *meta*

A **metamorphism** is useful to describe conversion from one data representation to another.

Example: *reformat*

Function *reformat n* is to process a text in *n*-character lines.

$$\begin{aligned} \textit{reformat} &:: \textit{Int} \rightarrow [[\textit{Char}]] \rightarrow [[\textit{Char}]] \\ \textit{reformat } n &= \textit{writeLines } n \cdot \textit{readLines} \end{aligned}$$

Specification with *meta*

A **metamorphism** is useful to describe conversion from one data representation to another.

Example: *reformat*

Function *reformat n* is to process a text in *n*-character lines.

$$\begin{aligned} \textit{reformat} &:: \textit{Int} \rightarrow [[\textit{Char}]] \rightarrow [[\textit{Char}]] \\ \textit{reformat } n &= \textit{writeLines } n \cdot \textit{readLines} \end{aligned}$$
$$\begin{aligned} \textit{readLines} &= \textit{concat} \\ \textit{writeLines } n &= \textit{unfold} (== []) (\textit{take } n) (\textit{drop } n) \end{aligned}$$

Specification with *meta*

A **metamorphism** is useful to describe conversion from one data representation to another.

Example: *reformat*

Function *reformat n* is to process a text in *n*-character lines.

$$\begin{aligned} \textit{reformat} &:: \textit{Int} \rightarrow [[\textit{Char}]] \rightarrow [[\textit{Char}]] \\ \textit{reformat } n &= \textit{writeLines } n \cdot \textit{readLines} \end{aligned}$$

$$\begin{aligned} \textit{readLines} &= \textit{concat} \\ \textit{writeLines } n &= \textit{unfold} (== []) (\textit{take } n) (\textit{drop } n) \end{aligned}$$

Exercise: L4-3

Give the evaluation result of
(*reformat* 3 [['l','a','m','h'], ['e','r'], ['e']]).

Specification with meta

Exercise: L4-6

The function *convert* is to convert a decimal to a binary. Suppose both decimal and binary is represented by a list of digits.

$$\text{convert } [1, 5] = [1, 1, 1, 1]$$

Define *convert* as a meta.